НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Комп'ютеризовані системи управління»
спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології»
на тему: «Система автоматизованого тестування»

Виконав (-ла):
студент (-ка) IV курсу, групи ІА-62
Вовчок Євген Миколайович _____

Керівник:
Аспірант кафелри Аутс
Вовк Євгеній Андрійович _____

Рецензент:
Доцент кафедри АСОІУ
Сперкач Майя Олегівна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 151 «Автоматизація та комп'ютерно-інтегровані технології»

Освітньо-професійна програма «Комп'ютеризовані системи управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____20__ р.

ЗАВДАННЯ
на дипломний проєкт студенту
Вовчка Євгена Миколайовича

1. Тема проєкту «Система автоматизованого тестування», керівник проєкту Вовк Євгеній Андрійович асистент, затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом проєкту _____

3. Вихідні дані до проєкту

4. Зміст пояснювальної записки Огляд існуючих рішень, пошук архітектурного рішення, пошук технічного рішення, розробка серверного додатка системи, розробка взаємодії серверного додатка з джерелами даних, реалізація концепції ‘Rest API’ і взаємодія клієнтської частини з серверним додатком, узгодження сервера з інтерфейсом користувача

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо) схема конкуруючих потоків, схема планувальника задач, архітектура системи.

6. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримати завдання на ДР	13 квітня 2020р.	
2	Аналіз предметної області	14 квітня 2020р.	
3	Огляд існуючих рішень	15 квітня 2020р.	
4	Пошук архітектурного рішення	16 квітня 2020р.	
5	Пошук технічного рішення	20-22 квітня 2020р.	
6	Розробка серверного додатка системи	22-31 квітня 2020р.	
7	Розробка взаємодії серверного додатка з джерелами даних	31 квітня 2020р.	
8	Реалізація концепції ‘Rest API’ і взаємодія клієнтської частини з серверним додатком	1-5 травня 2020р.	
9	Узгодження сервера з інтерфейсом користувача	5-10 травня 2020р.	
10	Тестування та налаштування міжсервісной взаємодії	10-15 травня 2020р.	
11	Опис роботи	15-21 травня 2020р.	
12	Оформлення текстової та графічної документації в межах формування звіту	1-10 червня 2020р	

Студент _____ Є. М. Вовчок
(підпис) (ініціали, прізвище)

Керівник роботи _____ Є. А. Вовк
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Вовчок Є. М. Система керування навігацією параплана. КПІ ім. Ігоря Сікорського, Київ, 2020.

Проект містить 65 с. тексту, 35 рисунки, 14 літературних джерел та 4 графічні матеріали.

Ключові слова: розробка програмного забезпечення, тестування, міжсервсіна взаємодія.

Об’єктом дослідження є тестування.

Предметом дослідження є система перерозподілу ваги питання.

Дипломний проект присвячено дослідженню та проектуванню системи, що забезпечує підвищення ефективності процедури оцінювання знань користувача. Принцип роботи системи засновано на дослідження не рівносильних питань. В індивідуальному проекті розроблено серверний веб додаток.

SUMMARY

Vovchok E.M. Paraglider navigation control system. KPI them. Igor Sikorsky, Kyiv, 2020.

The project contains 65 pages. text, 35 figures, 14 literature sources and 4 graphics.

Keywords: software development, testing, interservice interaction.

The object of research is testing.

The subject of the study is the system of redistribution of the weight of the issue.

The diploma project is devoted to the research and design of the system, which provides increased efficiency of the user knowledge assessment procedure. The principle of operation of the system is based on the study of non-equivalent issues. A server web application was developed in an individual project.

Пояснювальна записка
до дипломного проєкту
на тему: «Система автоматизованого тестування»

Київ – 2020 року

Номер рядка	Формат	Позначення	Найменування	Кільк. аркушів	Номер екзем.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІА62.050БАК.005 ПЗ	Пояснювальна записка	65		
6	А3	ІА62.050БАК.005 Э1	Структурна схема	1		
7	А3	ІА62.050БАК.005 Д1	Діаграма бази даних	1		
8	А3	ІА62.050БАК.005 Д2	Контекстна блок-схема	1		
9			роботи			
10	А3	ІА62.050БАК.005 Д3	Діаграма класів механізму	1		
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						
25						
26						
27						
28						

Зм.	Аркуш	№ докум.	Підпис	Дата	ІА62.050БАК.005 ТП		
Розроб.		Вовчок Є.М.			Система автоматизованого тестування	Літ.	Лист	Листів
Перевір.		Вовк Є.А.				Т	1	1
Реценз.					**Відомість технічного**	НТУУ «КПІ» ФІОТ		
Н. Контр.						Група ІА-62		
Затв.								

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

СУБД – система управління базами даних

ІТ – інформаційні технології

API - application programming interface

ІТС – інформаційно-телекомунікаційні системи

СУБД – система управління базами даних

ITIL- IT Infrastructure Library

ПЗ – програмне забезпечення

XML - eXtensible Markup Language

JSON - JavaScript Object Notation

SQL -Structured Query Language

БД – база даних

ANSI - American national standards institute

СУРБД - система управління реляційними базами даних

/LINQ - Language-Integrated Query

ВСТУП

В даний час використовується багато способів перевірки якості знань. Починаючи ще з школи нам пропонують такі контрольні заходи, як написання контрольних чи самостійних робіт, екзамени або ж захист предмету усно. Проте, протягом багатьох років, найпопулярнішим з них залишається – тестування.

Тестування - це не тільки одне із популярних, а і одне із найбільш різноманітних видів перевірки знань. Є багато типів тестування, проте можна виділити два основні типи – традиційне та адаптивне тестування.

Традиційний тест – це тест, який звикли бачити протягом навчання в університеті і на інших контрольних заходах по перевірці рівня знань. Він складає в собі набір запитань та відповіді на них, серед яких тільки одне чи декілька будуть вірними. Кожне питання в тесті, містить оцінку - це деяка кількість балів, яку називають ”вага запитання”. Результат після складання традиційного тесту – це набрана сума балів, що присвоюється за запитання, на які була дана правильна відповідь.

Адаптивний тест відрізняється від традиційного тим, що оцінювання відбувається по іншому. Після чого, уже не нараховується жодних балів, проте поділяєм питання за рівнем складності. Студент чи школяр починає проходити тест із найлегшого чи середнього по складності питання, і якщо його відповідь буде правильною – переходить до складнішого. Проте, якщо відповідь була невірною, перед ним буде питання, уже із нижчою складністю.

Також тестування залишається одним із основних видів перевірки знань, яке дуже складно оцінити субєктивно. У звязку з тим, що дуже часто виникають проблеми студентів із викладачами (вчителів з учнями), коли письмова чи усна розгорнута відповідь оцінюється достатньо субєктивно, так як немає точних критеріїв оцінювання “не точних” предметів. Тестування

вирішує цю проблему, зводячи відповідь до певного логічного висловлювання.

Проте, не кожний тест чи методика тестування забезпечує достовірні результати. На це може впливати багато факторів, про які, інколи, ніхто не задумується. Через це були складені вимоги до тестування студентів. Найважливішими з них є валідність, тобто дієвість тестування, надійність та справжня ефективність їх результатів. Саме ці критерії в вимогах до тестування використовують більшість вчених, спеціалізація яких є тестування.

В даний час є багато способів тестування, проте, дивлячись на критичну ситуацію яка сталась в даний час, набирає популярності автоматизоване тестування, яке можна проводити на будь-якій відстані, та яке забезпечує миттєвий результат.

Актуальностю даної роботи є використання інформаційних технологій для перевірки знань дозволяє індивідуалізувати навчальний процес, повністю забезпечує оперативний та точний самоконтроль і дозволяє здійснювати контроль з діагностикою похибок і з можливістю зворотного зв'язку. Існує безліч систем, що призначені для дистанційного навчання, які призначені для створення взаємодії між учнями та викладачем, та для проведення онлайн-тестування з функцією автоматичного результату, який миттєво обчислюється. У даній роботі було створену одну з таких систем, з додатковими можливостями, які забезпечують наявність власного кабінету та можливості перегляду історії тестування. У програмному додатку був представлений результат роботи програми.

Метою даної роботи є створення системи автоматизованого тестування, із додатковими функціями та зручного інтерфейсу використання.

Для досягнення поставленої мети були вирішені наступні завдання:

- огляд існуючих рішень систем аналізу та планування системи для оптимізації рівня знань;
- вивчення теорії по роботі з тестовими ситемами, їх структури та оцінювання;
- вибір можливих типів тестування;
- формування вимог до розроблюваної системи та вибір засобів розробки;
- розробка архітектури системи та її компонентів;
- створення додаткових механізмів;
- розроблення та оптимізація основного функціоналу системи за обраними методами;

Для створення даного проекту використано наступні засоби та технології розробки програмного забезпечення: .NET Framework, як одна з передових та зручних платформ для розробки програм; ASP.NET CORE 2.2, як розвинута підсистема сервісних веб-додатків, як об'єктно-орієнтовану технологію з більш високим рівнем абстракції на базі фреймворка .NET для роботи з даними; надійна, проста та надійна система керування базами даних MS SQL Server.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Системи автоматизованої перевірки знань, беруть свою історію розвитку із низки вдалих проектів, що в свою чергу починаються з наукових робіт В. П. Беспалька [1]. Дивлячись на його роботи, можна виділити декілька маштабних проектів. Серед них система компютерного автоматизованого тестування для медичних університетів. Також, дотепер в процесі розробки комбіновані системи, що вміщують у собі два види тестування, за допомогою інформаційних систем і одночасно з бланковим тестуванням.

До автоматизованого тестування відносяться всі ті ж вимоги, що я до бланкового. Отже, виділимо головні з них:

- валідність методу – називають збірну характеристику, що визначають процедури вимірювання та параметри засобу. Також її визначають властивості тієї ознаки, яка досліджується. Тобто, валідність методу – це достовірність того, що вимірюєм даним методом. Згідно з даною оцінкою, є можливість встановити сферу дії, для якої даний метод буде показувати статично правильні дані. Проте, якщо говорити про тести, які застосовуються для певного рівня знань чи кваліфікації, то рівень завдання, який задаєм у тесті, повинен відповідати рівню знань. Тобто, не можна пропонувати виконати завдання складнішого рівня завдання, якщо ваш рівень знань не відповідає йому;

  Щодо цього, говорять про тестову функціональну валідність.
- надійність тестування – або ж, як є можливість перефразувати надайність обраного методу вимірвання. Цим терміном позначається величина стійкості отриманих результатів, а це в свою чергу, впливає на точність, з якою вимірюєм певну ознаку. В першу чергу, віднолення даних результатів при кожних наступних вимірах стосується оцінки надійності тестування;

Рівень надійності технології може залежити насамперед від обєктивності методу та стабільних характеристиків які вимірюють, або ж параметр способу чи засобу вимірювання

Під даним параметром, розуміють рівень точності, за допомогою якого можем визначити ту, чи будь яку іншу ознаку. Отже, можна дійти висновку, що було визначено ступінь, наскільки є можливість покладатися на результати даного тесту. Про надійність та ефективність тестів деколи роблять рішення, за такими признаками: якщо при перевірці тесту, який був виконаний студентами, в результаті виявиться що згідно з показниками(оцінками), учні отримають ті ж результати, як і в інші рази, то даний тест є надійним.

Повертаючись до надійності тесту, потрібно зазначити, що вона також залежить не від інформаціїї що міститься в тестах та їх варіантах відповіді, а й від кількості запитань. Беручи до увагу дану інформацію, для надійності контролю знань доволі великих тем навчального курсу, тест повинен вміщувати в собі не менше, як 40 тестів.

- ефективність тестування

Вважається, що в силу початку нової теорії тестування, починається відчувати потребу розширення кількості вимог. Під третім критерієм, нам пропонують використовувати критерій ефективності.

Ефективність – під даним терміном, розуміємо порівняльний критерій, який дозволяє нам порівнювати між собою тести. Ефективним, назвається тільки той текст, за допомогою якого краще виміряти студенські знання, а саме: швидше; дешевше; знання виключно тих студентів, що підходять під потрібний рівень підготовки; якісніше; ніж за допомогою інакших тестів.

Якщо порівняти даний переметр ефективності із параметрами надійності та валідності, то найсуттєвіша відмінність буде саме у

переході від середнього до диференційованого показника. Давайте згадаєм, що надійність відносить до тесту, що складається з певної кількості запитань, які показують всім випробовуваним. Виключно в такому випадку є можливість знайти коефіцієнт надійності для тесту, як середню оцінку точності. Аналогічно в такий ж спосіб знаходимо валідність тексту. У випадку з ефективним тестом у нас все навпаки, так як даний тест, допускає відхід від середніх оцінок і від певної потрібної кількості завдань.

Однією з особливостей ефективних тестів, є дискримінативність.

До даних тестів також висувається декілька основних вимог, це розподіл за кількістю, складністю множени усіх тестових завдань.

- дискримінативність – визначення, що описує здатність окремих питань тесту. Дискримінативність вимірюється за допомогою показника дельта Фергюсона. Цей коефіцієнт – це зв'язок між значенням дискримінативності , яке було отримане для певного тесту і в даний момент є максимальним показником дискримінативності, що може показати такий тест;

Уважне конструювання питань тесту забезпечує відповідну оцінку дискримінативності, а зарахунок неї тести завжди виграють у інших форм випробувань.

Складність питань тесту – це певна характеристика завдання, яка забезпечує відображення статистичного рівня вирішуваності. Показником цієї характеристики є та частка людей, що проходили тестування і вирішили або не вирішили поставлене перед ними завдання. Для прикладу, якщо тільки 25% виконали завдання тесту, то його можна вважати складним, якщо 85% - легким, в межах даної категорії;

Підбір завдань для тестування за показником складності є дуже важливим для успішного застосування тестування. Тому що, при підборі занадто складних тестів валідність та надійність даного тесту буде різко зменшуватись. Проте, дуже прості завдання приведуть до неефективності тестування;

1.1. Автоматизоване тестування

Якщо порівнювати тестування з іншими видами контролю знань студентів, то тестування також має певні переваги та недоліки, із головних переваг, можна виділити:

- проведення тестування є об'єктивний та якісний спосіб оцінювання. Об'єктивність тестування досягається через стандартизацію процедури самого проведення тестів, а також перевірки якості тестів в цілому та завдань для них;
- тестування – це один з методів, який є "справедливим" по відношенню до студентів. Тести ставлять усіх студентів у одинакові умови, що стосується як процесу тестування так і оцінювання знань, що усуває субєктивну оцінку викладача. Згідно з статистикою учнів з Великої Британії, тести дозволяють зменшити кількість апеляційу три рази, а то і більше, та зробити дану процедуру одинаковою для всіх, незалежно від навчального закладу чи місця проживання;
- тестування – це дуже об'ємний інструмент, а саме через те, що воно може включати в себе тести по всіх темах із пройденого курсу, в той час як на письмовий чи усний екзамен зазвичай виноситься 2-5 тем. Даний інструмент дозволяє нам робити діагностику знань студента по всьому навчальному курсу, в той же момент виключаючи випадковість під час витягування білету або ж вибору теми;

Проте, також залишається деяка кількість недоліків та головні проблеми, серед яких можна виділити, такі як:

- якісний тестовий інструмент, а точніше його розробка – це досить трудомісткий, тривалий та затратний процес. Для більшості існуючих дисциплін, стандартний набір тестів ще не розроблений, а існуючі зазвичай мають дуже низьку якість;
- результат тестування, який отримує викладач, після проходження студентом тесту, включають дані про недостаток знань та підготовки, проте вони не показують причину цих недоліків;
- проведення тестування для творчих завдань, сюди ж відносяться і методологічні чи абстрактні, практичні знання;
- широке охоплення вивчених тем, які також описували у перевагах, має свої недоліки. Студенту, інколи, може бути недостатньо часу для підготовки;
- щоб забезпечити справделивості щодо проходження тестування, потрібно приймати спеціальні заходи, що забезпечують конфіденційність самх завдань. Тому, при повторних проходженнях тестів потрібно вносити зміни;
- тести можуть показувати не правильний результат, через випадкову помилку. Тобто той момент, коли студент правильно відповідає на складне запитання, проте у нього виникають труднощі із тими, що легші;

Були описані проблеми, що стосуються тестування в загальному.

Розглядаючи обрану тему, а саме “автоматизоване тестування” є можливість виділити ще дві головні проблеми, які нам потрібно буде вирішити перед переходом до проектування даної системи:

- спосіб створення запитань для тестів (У нас є можливість автоматизованого генерування тестових питань, де опираючись на основу математичних алгоритмів, або ж внесення їх вручну);

- інтерпритація результатів. На даний момент, в системах тестування вимірюється та фіксуються показники частки правильних відповідей(простота, корекція вгадування, за компонентами мережі), рівня трудності тестів, що студент виконує правильно з 50% імовірністю, та час виконання;

1.2. Генерування тестів. Алгоритми для тестових завдань

Виділяють два основних підходи для генерації тестів. Один з них, це генерація тестів що побудовано на основі параметризації. Даний спосіб дозволяє нам проводити тести багаторазово, так як кожного разу отримуються різні вибірки запитань.

Кожного разу, перед тим як почати програмну реалізацію даних завдань потрібно проводити математичне моделювання, для кожного з них. Нам потрібно розрахувати діапазони, де потрібно буде генерувати довільні величини параметрів, які входить в запитання.

Також, існує класичний підхід. Особливість даного методу в тому, що в його основі використовують двійку незалежних елементів: математичний алгоритм, що потрібний для розв’язування задачі на основі вхідної інформації та самого генератора, для вхідних даних. Під розв’язувальним пристроєм розуміють функцію, яка містить змінний набір вхідних параметрів, які в свою чергу залежать від вхідних параметрів. Автоматична генерація прийнятих даних, відбувається даним генератором за визначеними правилами. В свою чергу, дані правила описують зв’язки між початковими даними.

Дані, які були згенеровані за допомогою методу, який був описаний вище, подаються разом з шаблоном завдання та структурою XML формату на вхід до генератора. Після чого, генератор вставляє вже створену вхідну

інформацію в тест запитання, а отриману відповідь – до набору з іншими варіантами відповідей.

Також, даний метод має в основі реалізацію алгоритму, що дозволяє генерацію неправильних відповідей, звичайно, якщо цього вимагає тестове завдання. Вхідні дані або ж неправильні відповіді, про які говорили вище можуть випадковим чином обиратись із бази даних.

1.3. Задачі, які підлягають до автоматизації

Дискретна математика – це та галузь математики, яка вивчає дискретні структури, що виникають в застосуваннях та в межах математики. До цих структур, відносимо скінченний граф чи скінченну групу. Також сюди ж можуть відноситись математичні моделі структур-перетворювачів даних, наприклад: машина Тюринга, скінчені автомати. Проте, це приклади виключно скінченного характеру. Крім приведених вище структур, дискретна математика також містить у собі нескінченні графи, системи алгебри, певні обчислювані схеми, клітинні автомати та інше.

Дивлячись з боку доцільності, під час вибору задач, для системи автоматизації для початку потрібно генерувати саме такі завдання, на які вручну витратимо дуже багато зусиль та часу. Для таких завдань також існують характеристики:

Необхідність зображень. Виникають ситуаціїї, коли нам потрібно використовувати графи, дерева чи інші структури даних, а це потребує певних знань та навичок у використанні програмних засобів та потребує:

- велика кількість вхідних даних. Підбір коректних та узагальнених за певними категоріями даних є ресурсозатратною задачею;
- необхідність у великії кількості груп вхідних даних. Отримати куди більше різноманітних завдань дозволяє використання компютерних

засобів. Ці комбінації також можна зробити неповторюваними, і це буде куди ефективніше, ніж вручну;

- обчислення, що займає дуже багато часу порівнянням з поспосереднім введенням інофрмації користувачем. Результат тестування отримують моментально, і це забезпечує значну економію часу перевіряючого;

Характеристики, які описані вище можуть підходити під безліч задач, насамперед це задачі з деревами чи графами. Автоматизоване тестування при використанні автоматичного генерування похожих завдань дуже полегшує роботу викладача, зо економить час і відкидує потребу в установці додаткових програм.

1.4 Огляд існуючих рішень

Більшість наявних систем не об’єднують тестування та навчання шляхом встановлення незнання та його ліквідацію за допомогою опису і вказання правильних відповідей шляхом пояснень. Крім того ні одна з них не вирішує проблему еквівалентності ваги правильних відповідей. Також присутня проблема різноманітності форми тестових завдань і сучасності UX частини систем. Як приклад Moodle, мережева академія Cisco мають застарілий дизайн та вузький перелік форм тестових питань.

Після чого, було розглянуто платформу для вивчення мов програмувань GeekBrains [13]. Основним плюсами платформи є зручний та зрозумілий дизайн для звичайного користувача (рисунок 1.1).

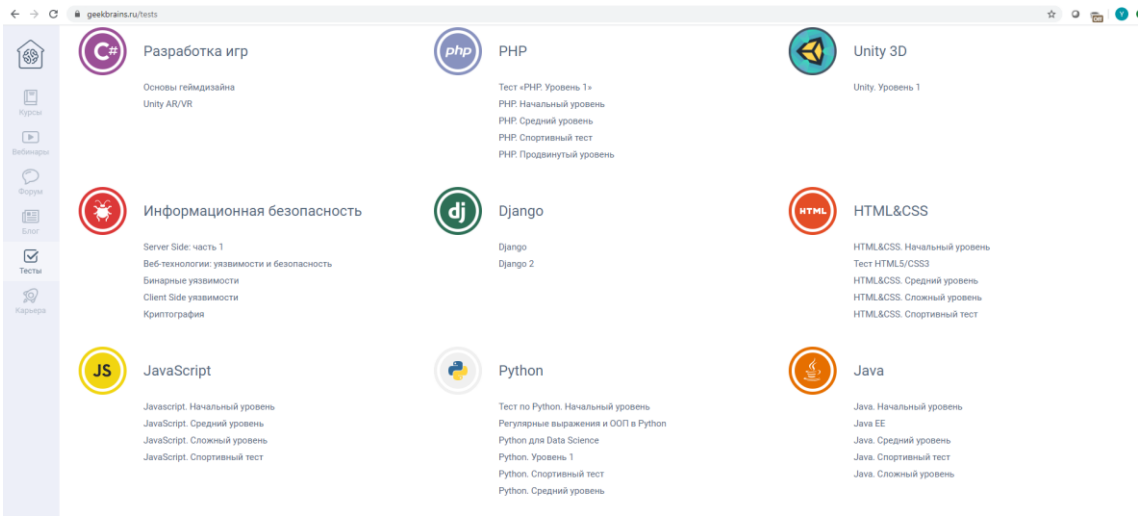

Рисунок 1.1. – Список категорій тестувань на платформі GeekBrains.

На рисунку 1.2, переходячи на вкладку “тести” відразу бачимо основні мови програмування та відповідно категорії з яких є можливість пройти тестування. Але в безкоштовній версій кількість тестів обмежена, що є мінусом системи.

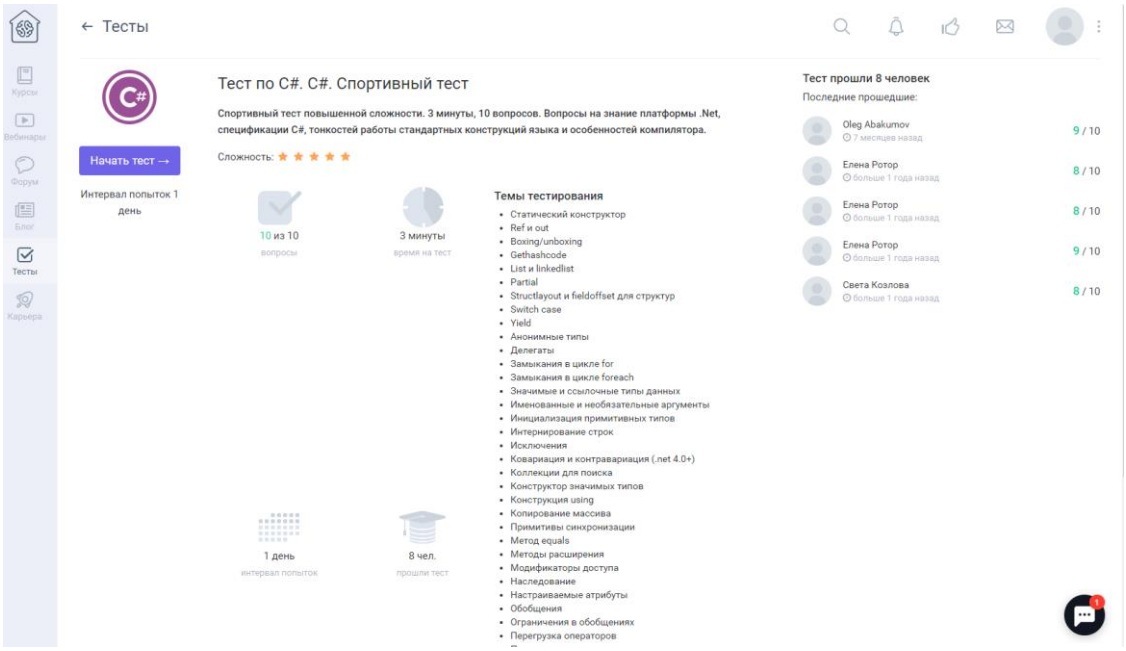

Рисунок 1.2. – Проходження тестування на платформі GeekBrains.

Було пройдено тест складної тяжкості по знанням платформи .NET. Перед тестом чітко отримали теми по якім будуть питання, кількість часу на

проходження та людей які проходили тест за сьогодні. В результаті проходження тесту було виявлено не рівносильні питання, що відносяться до різних рівнів знань, але вагу питання мають однакову це 10%. (Рисунок 1.3)

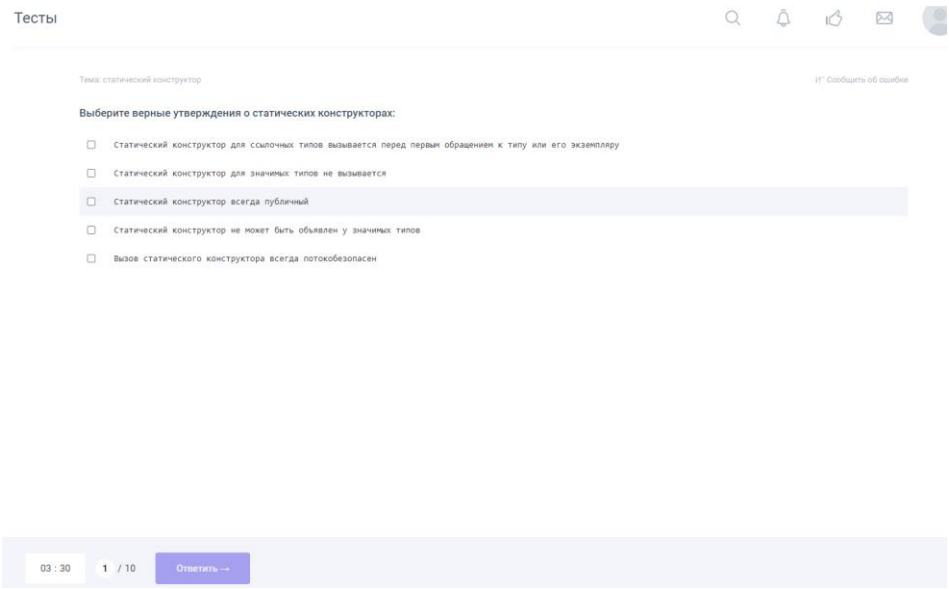

Рисунок 1.3. – Питання тесту про статичні конструктори.

Як видно з рисунку питання є відносно простим та базовим, хоча тяжкість тесту є максимальна. Наступне питання даного тесту. (Рисунок 1.4)

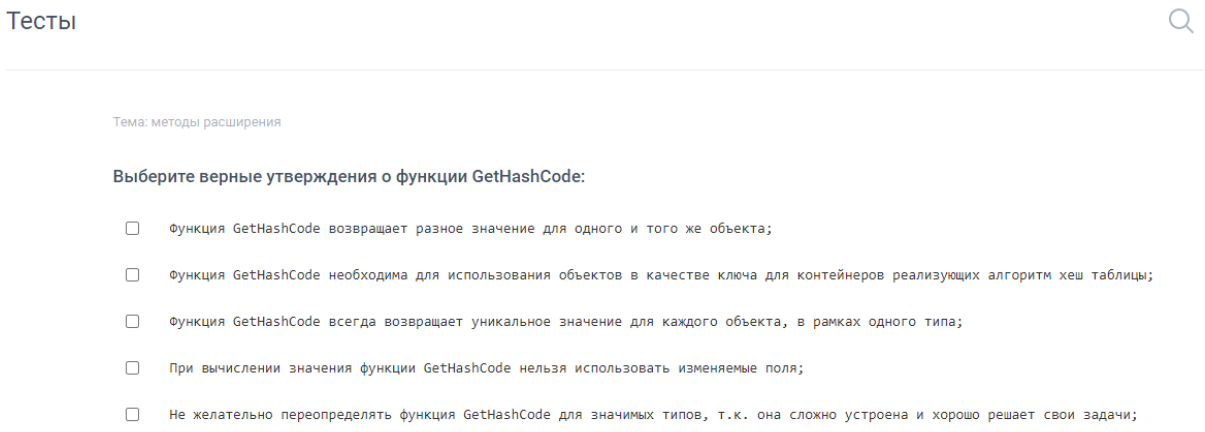

Рисунок 1.4. – Питання тесту про GetHashCode.

Питання про метод GetHashCode потребує відносно більше знань в архітектурі мови програмування C# та роботи структур даних. Згідно з цих

критерії питання про метод GetHashCode потребує більшої компетенції в знаннях платформи .NET ніж питання про статичні конструктори. Але оцінювання питання різної тяжкості оцінюється по 10%, що не дає об'єктивну оцінку про знання користувача, який проходить даний тест. Підсумують переваги та недоліки платформи GeekBrains.

Переваги:

- зручний та зрозумілий користувацький інтерфейс;
- велика кількість тестів з різних мов програмування;

Недоліки:

- у безкоштовній версії відносно невелика кількість тестів;
- в одному тесті нерівносильні питання, хоча оцінюються однаково;
- немає посилання на матеріали;

Розглянуто платформу для вивчення мов програмувань Quizful.net (Рисунок 1.5). Основним перевагами платформи є велика кількість тестів, але обмежена кількість тестів на день. Щоб зняти це обмеження, потрібно придбати підписку на поточний веб додаток.

Рисунок 1.5. – Список категорій тестувань на сайті Quizful.net.

Переходячи на сайт Quizful.net [14] відразу бачимо основні мови програмування та відповідно категорії з яких пройдене тестування. Але порівнюючи з платформою GeekBrains дизайн є застарілим. На рисунку 1.6 є можливість побачити проходження тестування на даній платформі.

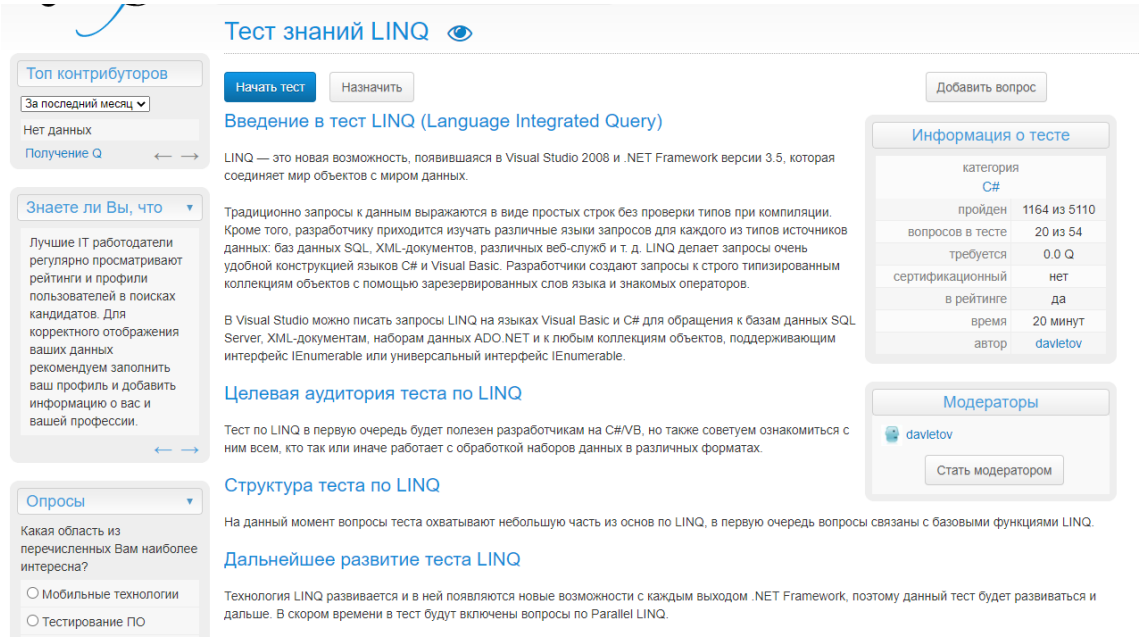

Рисунок 1.6. – Проходження тестування на платформі Quizful.net.

Було пройдено тест складної тяжкості по знанням платформи .NET теми LINQ. Перед тестом чітко отримали інформацію про питання, кількість часу на проходження та кількість людей які проходили тест за весь час. В результаті проходження тесту було виявлено не рівносильні питання, що відносяться до різних рівнів знань, але вагу питання мають однакову це 5%.

Рисунок 1.7. – Питання тесту про список та використання SelectMany.

Як видно з рисунку 1.7 питання направлено на знання роботи методу SelectMany, що є відносно зрозумілим і часто виокритованим під час розробки програмного рішення.

Рисунок 1.8. – Питання тесту про використання різних методів LINQ.

Виходячи з даного питання користувач, який проходить даний тест потрібно розуміти логіку роботи кілька LINQ методів, які використовуються в поточному питані. Порівнюючи з першим питанням є відносно тяжким оскільки потребує набагато більшої компетенції в знанях про LINQ (Рисунок 1.8). Але оцінювання питання різної тяжкості оцінюється по 5%, що не дає об’єктивну оцінку про знання користувача, який проходить даний тест. Підсумовуючи переваги та недоліки платформи Quizful.net:

Переваги:

- велика кількість тестів з різних мов програмування;

Недоліки:

- у безкоштовній версії обмежена кількість тестів за одну добу;
- застрілий дизай;
- в одному тесті нерівносильні питання, хоча оцінюються однаково;
- немає посилання на матеріали;

В результаті огляду прийнято наступні вимоги до проектованого рішення:

- вирішення проблеми еквівалентності ваги правильних відповідей шляхом впровадження динамічного показника відносного загальній статистиці відповідей на питання тесту;
- можливість додавання пояснень та посилань до відповідних питанням тем та їх пов’язування;
- розробка сучасного дизайну;
- впровадження різноманітних форм тестових питань з можливістю конфігурації;
- широка спеціалізація системи, шляхом налаштувань формату тестів;

1.5 Формулювання індивідуального завдання на розробку

В процесі аналізу предметної області було виділено основні процеси контролю знань:

- формування питань для контролю знань на основі контрольних завдань, що зберігаються до джерела даних, користувач обирає правильні варіанти зі списку;
- видача їх користувачу та отримання його відповіді, можливо, зі зворотним зв'язком;
- виставлення оцінки;

. Для повної реалізації функціональних вимог і конкурентоздатності системи з переліку було обрано як ті, що підлягають імплементації, наступні види тестів:

Коротка відповідь – користувач повинен ввести правильний варіант відповіді в текстове поле. Для правильної відповіді необхідно добре розбиратись у темі питання.

Правильно/неправильно – користувач повинен визначити чи правдиве твердження в питанні. Це найпростіший тип питання.

Вибір однієї відповіді – користувачу потрібно вибрати один правильний варіант з запропонованих варіантів.

Вибір кількох відповідей – користувач обирає правильні варіанти відповіді зі списку. Тести такого типу складніші чим вибір однієї відповіді, так як кількість правильних відповідей заздалегідь невідома. Правильна відповідь методом випадкового підбору малоймовірна.

Послідовність – користувач розміщує елементи у правильній послідовності

Числова відповідь – користувач вводить число у поле для відповіді. Вгадати правильну відповідь малоймовірно.

Відповідність – користувач повинен з’єднати пари, що складаються з слів, зображень чи фраз. Додаткові «зайві» варіанти відповідності можуть ускладнити питання.

При створенні тесту є необхідність виставлення прохідного балу. Крім того існує проблема еквівалентності ваги правильних відповідей, оскільки різні питання можуть посилатися на різні об’єми знань, можуть мати неточності, бути складними для розуміння. Універсальний рецепт для фіксованого значення прохідного балу відсутній, тому необхідно зробити його відносним. Як рішення для забезпечення об’єктивності тестування обрано динамічне визначення ваги правильних відповідей відносно загальної статистики відповідей на питання тесту.

2 АРХІТЕКТУРА ДОДАТКУ

2.1. Централізована система клієнт – сервера

Архітектура програмного забезпечення - це процес перетворення таких характеристик програмного забезпечення, як: гнучкість, масштабованість, можливість реалізації, багаторазовість використання і безпека в структуроване рішення, яке відповідає технічним і функціональним вимогам.

Програмне забезпечення повинно «легко розширювати свій функціонал, складатися з блоків і бути легким в обслуговуванні».

Рисунок 2.1. – Схема клієнт-серверного рішення.

Зображена на рисунку 2.1, модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером. Логічно можна відокремити три рівні операцій:

- Рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;
- Прикладний рівень, який реалізує основну логіку застосунку і на якому здійснюється необхідна обробка інформації;
- Рівень управління даними, який забезпечує зберігання даних та доступ до них.

Виходячи з поточного розділення, є можливість виокремити основні “плюси” і “мінуси” архітектурного клієнт - серверного підходу.

Резюмують плюси архітектури:

- потужний сервер є дешевше відносно великої кількості потужних клієнтських машин;
- немає дублювання коду - основний код зберігається на сервері;
- персональні дані в безпеці - простий користувач не бачить зайвого;

Мінуси архітектури:

- якщо сервер є не доступний через технічні проблеми, то клієнти не зможуть користуватися програмним додатком;

Саме тому в програмних додатка архітектуру ускладнюють і навіть дублюють. Але в таких випадках клієнтська сторона не розуміє. Куди направляти відповідний запит. В таких випадках перед сервером ставлять додатковий механізм, який називається “Балансувальник”. Тепер не є важливим скільки буде серверів для клієнта бо всі запити клієнт відправляє тільки на балансувальник (Рисунок 2.2).

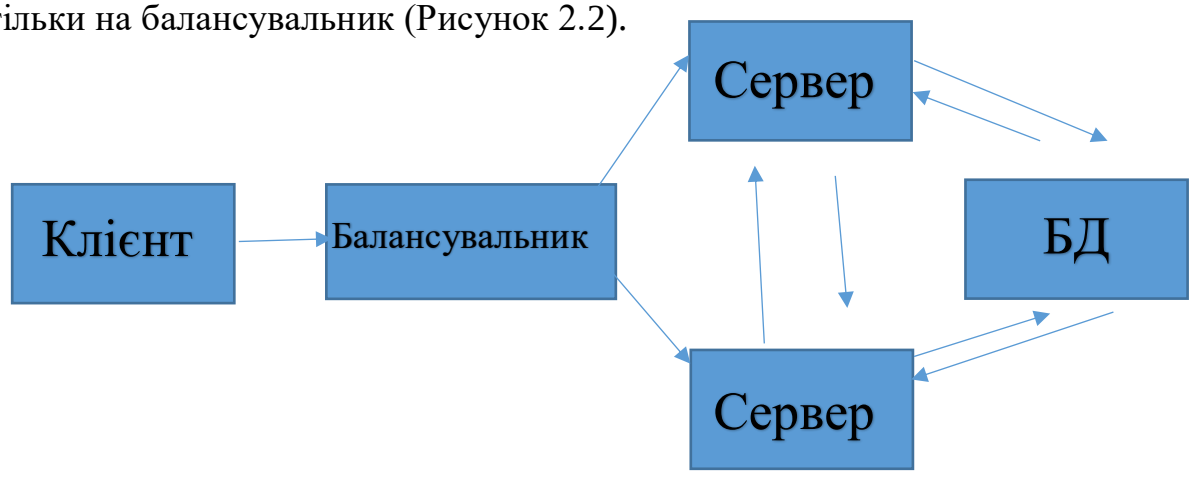

Рисунок 2.2. – Схема клієнт-серверного рішення з балансувальником.

2.2. Авторизація та регістрація користувачів

На процесах аутентифікації і авторизації засновано поділу прав доступу, без якого не обходиться жодний веб застосунок. Перед тим, як перейти до технічних деталей, виділимо основні аспекти:

- ідентифікація - це заява про те, ким користувач є. Залежно від реалізації, це може бути, адреса електронної пошти, номер облікового запису, номер телефону, тощо;
- аутентифікація - надання доказів, що користувач насправді є той, ким ідентифікувалися;
- авторизація - перевірка, що користувачу дозволено доступ до поточно веб ресурсу;

При використанні HTTP-протоколу найпростіший спосіб аутентифікації – “Basic access authentication”. Цей протокол є не актуальним і вже відносно рідко майже не використовується в мережі інтернет, особливо в незахищених з'єднаннях, але ще застосовується у внутрішньо корпоративних системах, просто тому що деякі з веб застосунків створені досить давно. Стосовно до веб-застосунків даний тип аутентифікації працює наступним чином:

- сервер, при зверненні неавторизованого користувача до ресурсу, направляє запит із статусом "401 Unauthorized" і додає заголовок "WWW-Authenticate" із зазначенням параметрів аутентифікації;
- браузер, при отриманні поточної відповіді, автоматично генерує діалог введення ім’я користувача і пароля. Користувач вводить деталі свого облікового запису;
- у всіх наступних запитах до поточного веб-сайту браузер автоматично додає заголовок "Authorization", в якому передаються певні дані клієнта для аутентифікації сервером;

- сервер аутентифікує клієнта за даними з поточного HTTP заголовка. Механізм про надання авторизація виконується окремо згідно з роллю користувача, або інших даних облікового запису;

Весь процес стандартизований і підтримується всіма відомими браузерами і веб-серверами. Існує кілька схем аутентифікації, що відрізняються за рівнем безпеки:

- basic – відносно проста схема, при якій ім'я користувача і пароля користувача передаються в заголовку HTTP в незашифрованому вигляді (Рисунок 2.3);

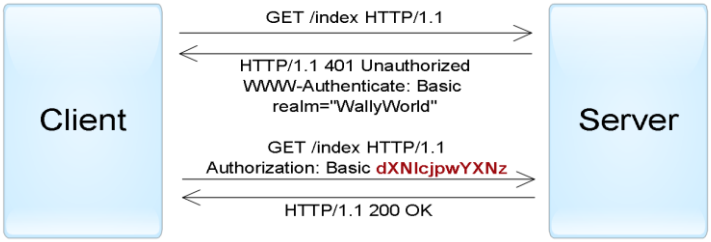

Рисунок 2.3. – Аутентифікації з використанням Basic схеми.

- digest – це схема запит-відповідь, коли сервер призначив унікальне значення для одноразового номера, а браузер передав MD5-хеш пароля, обчислений з використанням зазначеного одноразовго номера. Це є більш безпечна альтернатива Basic схеми при не безпечних з'єднаннях, але є вразливою до атак. Крім того, застосування цієї схеми не дозволяє реалізувати сучасні хеш-функції для зберігання паролів на сервері;
- NTLM (відомий як автентифікація Windows) також заснований на підході-відповіді, в якому пароль не передається в чистому вигляді. Ця схема не є стандартом HTTP, але підтримується більшістю браузерів та веб-серверів. В основному використовується для аутентифікації користувачів Windows Active Directory у веб-додатках. Вразливий до передач хеш-атак;
- negotiate - це ще одна схема сімейства аутентифікації Windows, яка дозволяє клієнту вибирати між аутентифікацією NTLM і Kerberos.

Kerberos - відносно безпечний протокол єдиного входу. Однак він може функціонувати тільки в тому випадку, якщо і користувач, і сервер знаходяться в зоні інтрамережі і є частиною домену Windows;

Також потрібно відмітити, що використання такого типу аутифікації у користувача немає можливості вийти з веб-застосунка, але користувач може закрити браузер, що не є ефективним використанням веб-додатка для користуача.

З наступних видів аутифікації є Forms authentication (Рисунок 2.4). Для поточного протоколу немає єдиного стандарту і всі реалізації є різними для кожної системи. Даний проточкол працює наступним чином:

- у веб застосунок влаштовується HTTP – форма, в яку користувач вносить свої дані, наприклад це може бути ім’я користувача і пароль;
- поточно форма відправляється на сервер для аутифікації;
- у випадку успішної аутифікації веб застосунок створює “session token”, який зазвичай знаходитсья в “ browser cookies ”;
- при наступних запитів браузера “session token” автоматичноно передається на сервер і це дозволяє отримати всю інформацію про користувача;

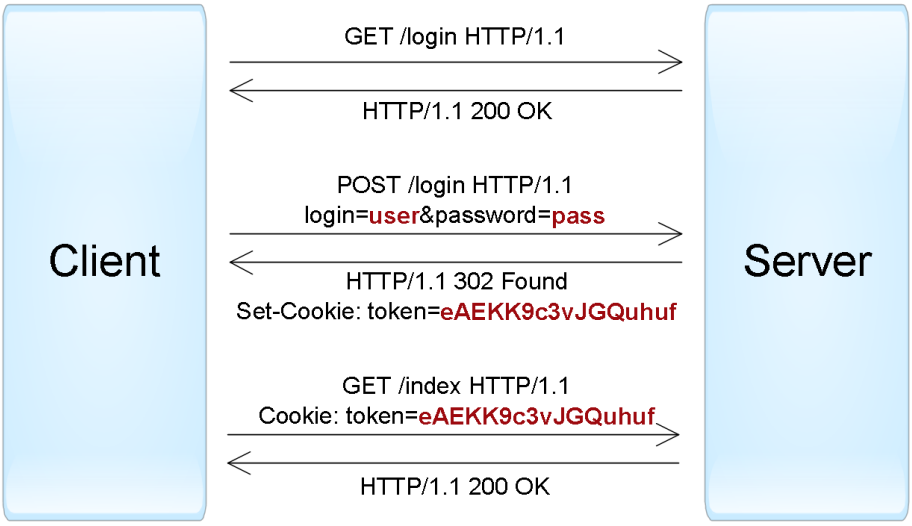

Рисунок 2.4. – Схема аутифікації Forms authentication.

Аутентифікація по токенам найчастіше застосовується при побудові розподілених систем єдиного входу (SSO), де один додаток (постачальник послуг або покладається сторона) делегує функцію аутентифікації користувача іншій програмі (постачальнику ідентифікаційних даних або службі аутентифікації). Типовим прикладом цього методу є вхід у додаток через обліковий запис у соціальних мережах. Тут соціальні мережі є послугами аутентифікації, а додаток довіряє користувачам функції аутентифікації в соціальних мережах.

Реалізація цього методу полягає в тому, що постачальник посвідчень (IP) надає надійну інформацію про користувача в формі токена, а додаток постачальника послуг (SP) використовує цей токен для ідентифікації, аутентифікації і авторизації користувача.

В цілому весь процес виглядає наступним чином:

- клієнт проходить аутентифікацію у провайдера ідентифікації одним зі специфічних для нього способів (пароль, пароль, сертифікат, Kerberos і т. Д.) ;
- клієнт просить постачальника посвідчень надати йому токен для конкретного додатка SP. Постачальник посвідчень генерує токен і відправляє його клієнту;
- клієнт аутентифицирующей в додатку SP, використовуючи цей токен;

Існує кілька поширених форматів токенов для веб-додатків:

- Simple Web Token (SWT) - найбільш простий формат, який представляє собою набір довільних пар ім'я / значення в форматі кодування HTML форми;

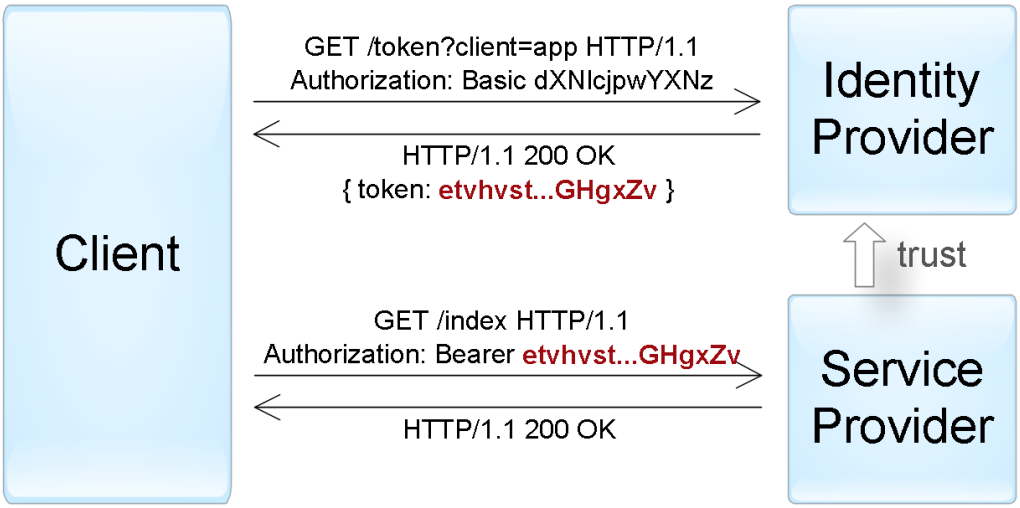

Рисунок 2.5. – Схема аутифікації по токенам.

- JSON Web Token (JWT) - містить три блоки, між якими ставлять крапку: заголовок, набір полів (claims) і підпис. Перші два блоки представлені в JSON-форматі і додатково закодовані в формат base64 (Рисунок 2.5);

2.3. Математична модель алгоритму

Основна модель алгоритму включає алгоритм ранжування.

Ранжування - завдання сортування набору елементів з міркування їх релевантності. Найчастіше релевантність розуміється по відношенню до деякого об'єкту. Наприклад, в завданню інформаційного пошуку об'єкт - це запит, елементи - всілякі документи (посилання на них), а релевантність - відповідність документа запиту, в завданню рекомендацій же об'єкт - це користувач, елементи - той чи інший рекомендований контент в даному випадку вага запитання, а релевантность - ймовірність того, що користувач виконає поточний тест даними контентом.

Формально, N об'єктів $U = \{u_i\}_{i=1}^{N}$ і M елементів $E = \{u_j\}_{j=1}^{M}$. Результат роботи алгоритму ранжирування елементів E для об'єкта $u \in U$ - це відображення $r : E \to R$, яке зіставляє кожному елементу

$e \in U$ вага $r(e)$, що характеризує ступінь релевантності елемента об'єкту (чим більше вага, тим релевантні об'єкт). При цьому, набір ваг $\{r(e)\}_{e \in E}$ задає перестановку $\pi : [1..M] \to [1..M]$ на наборі елементів елементів E (вважають, що безліч елементів впорядковане) виходячи з їх сортування по спадаючій ваги $r(e)$.

Щоб оцінити якість ранжирування, необхідно мати певний «еталон», з яким можна було б порівняти результати алгоритму. $r^{true}: E \to [0,1]$ - еталонну функцію релевантності, що характеризує «справжню» релевантність елементів для даного об'єкта ($r^{true} = 1$ - елемент ідеально підходить, $r^{true} = 0$ - повністю нерелевантний), а також відповідну їй перестановку $\pi^{true}: E \to [0..M]$ (по спадаючій $r^{true}(e)$).

Існує два основних способи отримання r^{true}:

На основі історичних даних. Наприклад, в разі рекомендацій контенту, можна взяти перегляди тестів користувача і присвоїти переглянутих ваг відповідних елементів 1 $r^{true}(e) \leftarrow 1$а всім іншим – 0. На основі експертної оцінки. Наприклад, в задачі пошуку, для кожного запиту можна залучити команду асессоров, які вручну оцінять релевантності документів запиту;

Варто зазначити, що коли r^{true} приймає тільки екстремальні значення: 0 і 1, то престановку π^{true} зазвичай не рассматрівют і враховують лише безліч релевантних елементів, для яких $r^{true} = 1$.

Мета метрики якості ранжирування - визначити, наскільки отримані алгоритмом оцінки релевантності $r(e)$ і відповідна їм перестановка π відповідають істинним значенням релевантності r^{true}.

2.4. Технічні рішення

Розробки технічного рішення для серверної частини, найпопулярніші технології для побудови веб застосунків, а саме CGI (C++), ASP.NET CORE (C#) та Spring (Java).

Порівнюється кожна з поточних технологій та описуються переваги і недоліки реалізації веб-додатків (Рисунок 2.6). Перше за все порівнюється CGI (C++) і ASP.NET CORE (C#) за допомогою однакового запиту сервером, а саме швидкість пошуку та видачі даних на порівнюваних платформах

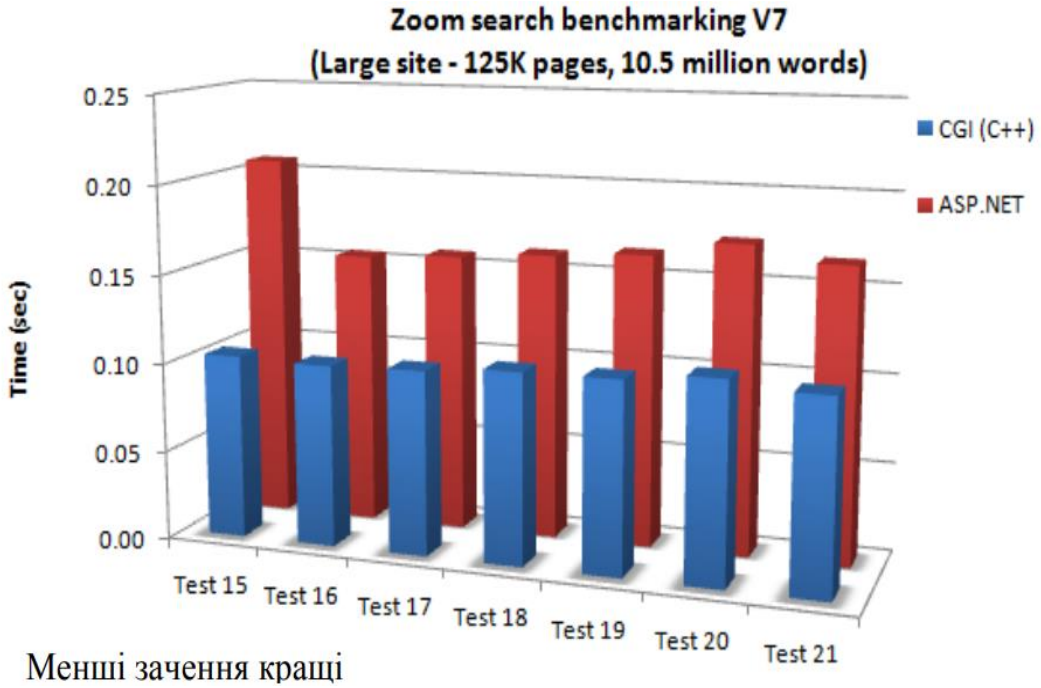

Рисунок 2.6. - Порівняння швидкодії реалізацій серверних рішень CGI та ASP.NET CORE.

Виходячи з даних графіку представленого на рисунок. 2.6., рішення на базі CGI має більшу швидкодію. Проте, на відносно невеликих об’ємах даних розрив у швидкодії скорочується. Крім цього розробка на базі мови C++ є значно витратнішою та займає більше часу, через технічні особливості мови та наявність безпосереднього управління оперативною пам’яттю в межах створюваного процесу[1].

В межах подальшого огляду було проведено порівняння ряду конфігурацій систем ASP.NET CORE та системи на базі Spring [8], результати якого представлено на рисуноку 2.7.

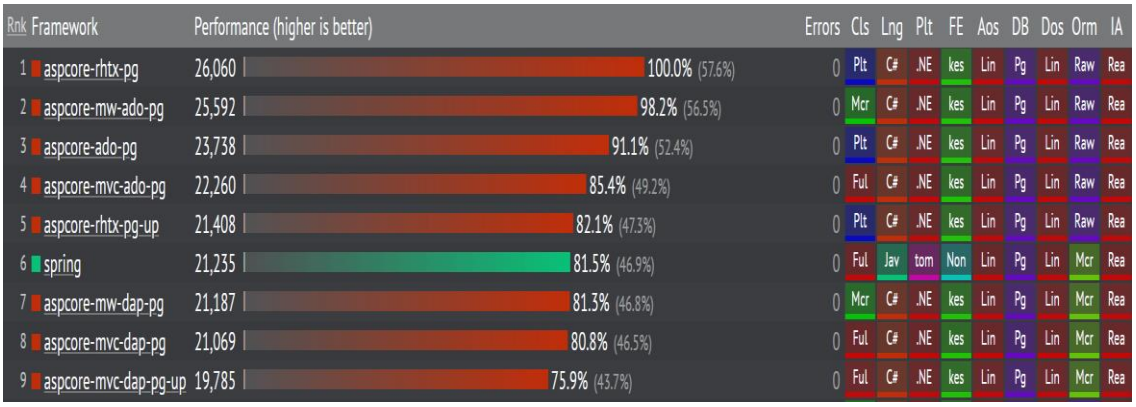

Рисунок 2.7. - Порівняння швидкодії різних конфігурацій систем на базі ASP.NET CORE та системи на базі Spring.

У результаті порівняння були зроблені висновки, що для розробки об’ємного проекту у якості основи найкраще підходить платформа .NET, технолоія ASP.NET CORE 2.2. Він забезпечує максимальну гнучкість та велику завантаженість за рахунок асинхронності. Інфраструктура ASP.NET MVC Framework реалізує шаблон MVC і при цьому забезпечує істотно поліпшена поділ відповідальності. Насправді в ASP.NET MVC впроваджений сучасний варіант MVC, який особливо добре підходить для веб-додатків.

За рахунок прийняття та адаптації шаблону MVC інфраструктура ASP.NET MVC Framework становить сильну конкуренцію Java чи Node.js і аналогічним платформ, виводячи модель MVC в авангард розвитку світу .NET. Узагальнюючи досвід і найбільш рекомендовані прийоми, виявлені розробниками, які використовують інші платформи, ASP.NET MVC у багатьох відношеннях перевершила навіть те, що може запропонувати Rails.

Інфраструктура MVC Framework побудована у вигляді набору незалежних компонентів, які задовольняють інтерфейсу .NET або створені на основі абстрактного базового класу. Компоненти, подібні системі

маршрутизації, механізму візуалізації і фабриці контролерів, можна легко замінювати іншими компонентами з власної реалізацією. У загальному випадку для кожного компонента MVC Framework пропонує три можливості:

- використання стандартної реалізації компонента в тому вигляді, як вона є (цього повинно бути достатньо для більшості додатків) ;
- створення підкласу зі стандартної реалізації з метою коригування існуючого поведінки;
- повна заміна компонента нової реалізацією інтерфейсу або абстрактного базового класу;

2.4.1. Проектування доступу до даних

Існує відносно багато технології які дозволяють зручно та швидко працювати з базами даними, яка була спроектована індивідуальним завдання дипломного проекту іншим учасником. Оскільки вибір було зроблений SQL бази даних, то відповідно було розглянуто наступні технології по роботі з реляційні системи управління базами даних:

- ADO.NET;
- Entity Framework Core (EF Core);

У ADO.NET використовується багаторівнева архітектура, яка обертається навколо невеликого числа ключових концепцій, таких як об'єкти Connection, Command і DataSet.

Постачальник даних (data provider) - це набір класів ADO.NET, які дозволяють отримувати доступ до певної базі даних, виконувати команди SQL та видавати дані. По суті, постачальник даних - це міст між додатком і джерелом даних.

У першому наближенні постачальник даних можна розглядати як набір типів, визначених в даному просторі імен, який призначений для взаємодії з конкретним джерелом даних. Однак незалежно від використовуваного

постачальника даних, кожен з них визначає набір класів, що забезпечують основну функціональність. У таблиці нижче наведені деякі загальні основні об'єкти, їх базові класи (визначені в просторі імен System.Data.Common) і основні інтерфейси (визначені в просторі імен System.Data), які вони реалізують.

Конкретні імена цих основних класів розрізняються у різних постачальників (наприклад, SqlConnection, OracleConnection, OdbcConnection і MySqlConnection), але всі ці об'єкти породжені від одного і того ж базового класу (в разі об'єктів підключення це DbConnection), який реалізує ідентичні інтерфейси (на кшталт IDbConnection) . Тому якщо ви навчитеся працювати з одним постачальником даних, то легко впораєтеся і з іншими.

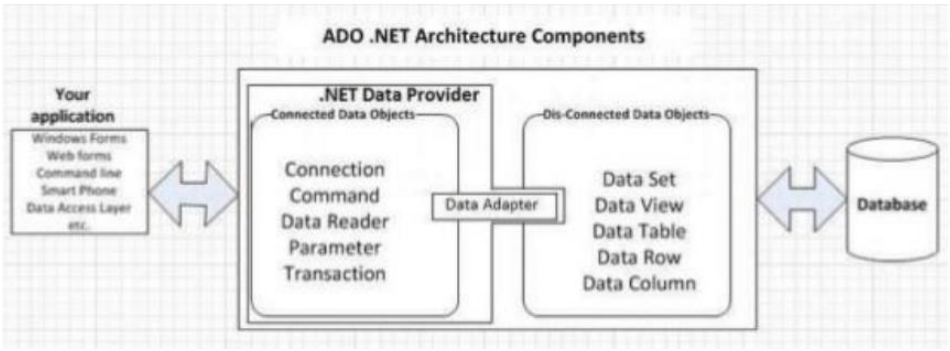

Рисунок 2.8. - ADO.NET Архітектура.

У ADO.NET термін "об'єкт підключення" насправді відноситься до конкретного типу, породженому від DbConnection; об'єкта підключення "взагалі" немає. Те ж можна сказати і про "об'єкті команди", "об'єкті адаптера даних" (Рисунок 2.8). За угодою імена об'єктів в конкретному постачальнику даних мають префікси відповідної СУБД (наприклад, SqlConnection, OracleConnection, SqlDataReader і т.д.).

Однією з ключових ідей, що лежать в основі моделі постачальників ADO.NET, є розширюваність. Іншими словами, розробники можуть створювати власні постачальники для патентованих джерел даних. Насправді є безліч підтверджують це прикладів, які демонструють, як створювати власні

постачальники ADO.NET, службовці оболонками для нереляційних сховищ даних, таких як файлова система або служба каталогів. Деякі незалежні виробники також продають власні постачальники даних для .NET.

Схема, яка зображує основні компоненти (і способи взаємодії з ними) Entity Framework CORE (Рисунок 2.9).

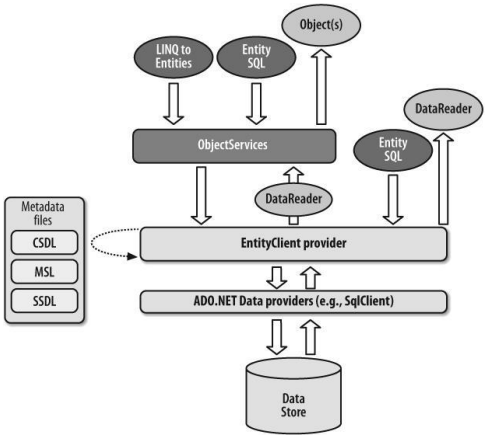

Рисунок 2.9. - Архітектура Entity Framework CORE.

Робота EF виглядає приблизно наступним чином:

Формується запит до EntityClient (за допомогою Object Services або безпосередньо)

EntityClient за допомогою метаданих перетворює Entity SQL або Linq To Entities-запит в SQL-запит

SQL-запит віддається на виконання ADO.NET провайдеру, вказаною в рядку з'єднання (в даному випадку - SqlClient)

Після звернення провайдера до СУБД дані в зворотному порядку повертаються. Три основних (на мій погляд) компонента архітектури EF: Metadata Files (EDM), Entity Client і Object Services.

EntityClient - самий низькорівневий спосіб створення запитів до EDM. EntityClient відрізняється від Object Services тим, що він не матеріалізує дані в об'єкти, а просто повертає дані у вигляді рядків і стовпців за допомогою EntityDataReader.

На практиці найпопулярнішим застосуванням EntityClient є реалізація того, що не вміє (або вміє, але погано) Object Services (наприклад, робота з деякими типу збережених процедур).

Object Services є верхівкою API Entity Framework CORE і розташовується в просторі імен System.Data.Objects. Object Services реалізує весь необхідний функціонал для зручного створення і взаємодії з сутностями концептуальної моделі. Основним компонентом Object Services є клас ObjectContext, спадкоємцем якого в даній моделі є згенерований дизайнером клас FirstModel.

Виходячи з описаних технологій оптимальним вибором є Entity Framework CORE на основі наступних критеріїв:

- наявність підтримки способів генерації бази даних CodeFirst, DataBaseFirst;
- наявність можливості написання запитів засобами мови LINQ та їх подальше виконання всередині бази даних SQL;
- містить генерацію бази даних з відповідно заданої об’єктно-орієнтованої моделі;
- наявність механізму кешування даних;

2.4.2. Моделювання поведінки системи

Основним компонентом, як будь-якого іншого сервісного додатку являється бізнес-логіка. Основні задачі бізнес-логіки це переважно обробка даних, отриманих з рівня доступу до даних, використанням поведінкових шаблонів, які переважно описують логіку оброблення даних. Компоненти такої системи вже реалізує кінцеву функціональність, яку очікують отримати. Одним з відомих патернів, який розмежовує зовнішній вигляд і логіки поведінки з кожної сторони, являється MVC (Model-view-controller). Детальніше про даний патерн на основі платформи .NET, нижче.

Model-view-controller (MVC, «Модель-уявлення-контролер») – це патерн проектування, використовується для розділення користувацького інтерфейса (уявлення), даних (модель) і логіки серверу (контролер). Згідно з таким розділенням, патерн MVC надає змогу модифікувати одного з компонентів з мінімальними змінами на інші (Рисунок 2.10).

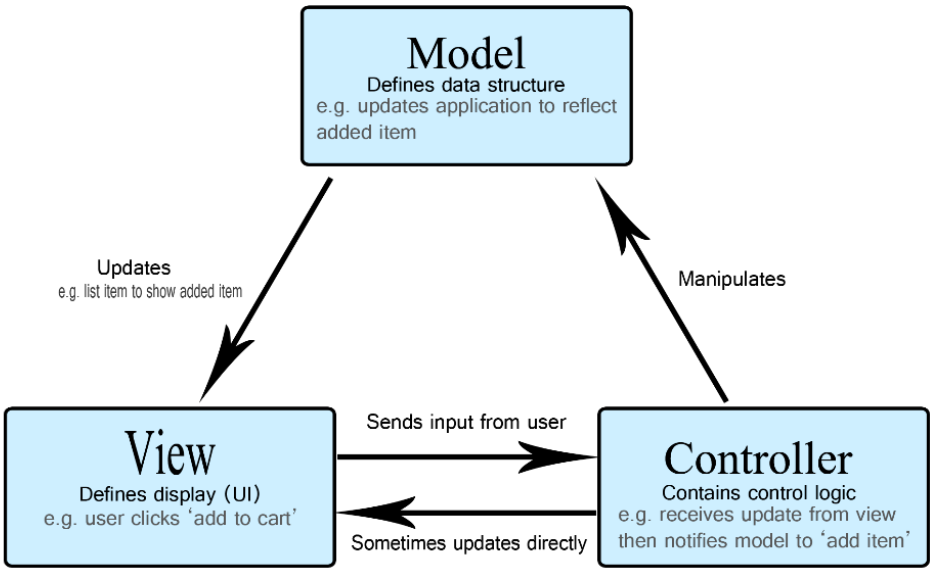

Рисунок 2.10. – Схема патерну MVC.

2.4.3. Порівняння планувальників задач

Згідно з поставленим індивідуальним завданням потрібно вибрати планувальник задач. Популярні планировщики задач для .NET, а саме:

- Hangfire;
- Quartz.NET;

Hangfire - багато-і масштабований планувальник завдань, побудований за клієнт-серверній архітектурі на стеку технологій .NET (в першу чергу Task Parallel Library і Reflection), з проміжним зберіганням завдань в БД. Повністю функціональний в безкоштовній (LGPL v3) версії з відкритим вихідним кодом.

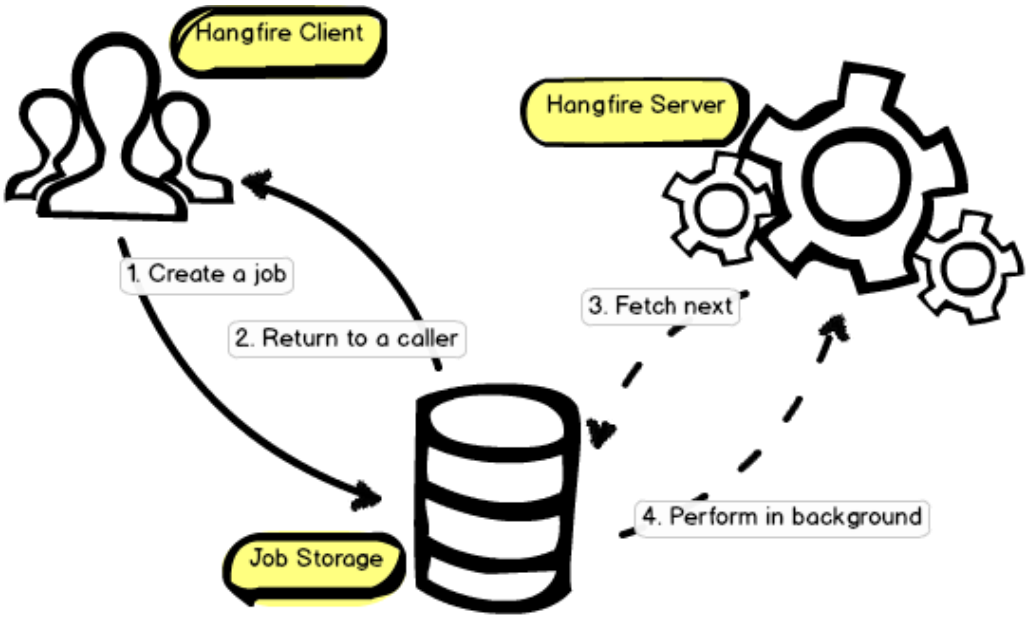

Рисунок 2.11.– Процес роботи Hangfire

Згідно з рисунку 2.11, процес-клієнт додає завдання в БД, процес-сервер періодично опитує БД і виконує завдання. Важливі моменти:

- все, що пов'язує клієнта і сервера - це доступ до загальної БД і загальним збірок, в яких оголошено класи-завдання;
- масштабування навантаження (збільшення кількості серверів) ;
- без сховища завдань Hangfire не працює і працювати не може. За замовчуванням підтримується SQL Server, є розширення для ряду популярних СУБД;
- в якості хоста для Hangfire може виступати що завгодно: ASP.NET-додаток, Windows Service, консольний додаток і т.д. аж до Azure Worker Role;

Quartz.NET вирішує схожі завдання, як і Hangfire - підтримує довільну кількість «клієнтів» (додавання завдання) і «серверів» (виконання завдання), що використовують загальну БД. Але виконання різний.

Поділу на клієнтську і серверну частину в проекті немає - Quartz.NET поширюється у вигляді єдиної DLL. Для того, щоб конкретний екземпляр

додатка дозволяв тільки додавати завдання, а не виконувати їх - необхідно його налаштувати.

Quartz.NET повністю безкоштовний, «з коробки» пропонує зберігання завдань як in-memory, так і з використанням багатьох популярних СУБД (SQL Server, Oracle, MySQL, SQLite і т.п.). Зберігання in-memory є по-суті звичайний словник в пам'яті одного єдиного процесу-сервера, що виконує завдання. Реалізувати кілька процесів-серверів стає можливим тільки при збереженні завдань в БД. Для синхронізації, Quartz.NET не покладається на специфічні особливості реалізації конкретної СУБД (ті ж Application Lock в SQL Server), а використовує один узагальнений алгоритм. Наприклад, шляхом реєстрації в таблиці QRTZ_LOCKS гарантується одноразова робота не більше ніж одного процесу-планувальника з конкретним унікальним id, видача завдання «на виконання» здійснюється простим зміною статусу в таблиці QRTZ_TRIGGERS.

Що стосується многопоточності, то Quartz.NET використовує класи з простору імен System.Threading: new Thread () (див. Клас QuartzThread), свої пули потоків, синхронізація через Monitor.Wait / Monitor.PulseAll.

Виходячи з цього для поставленої задачі було принято використовувати планувальник завдань Hangfire.

3 ПРОГРАМНЕ РІШЕННЯ

3.1 Загальна структура бізнес логіки

Згідно з вище описаним підходом аналізу програмної області, було визначено, що запропоноване програмне рішення буде містити три-рівневу ахітектуру, яка в свою чергу складається з частини користувача, що побудована як окремий проект, який взаємодіє з кінцевим API серверу. Виходячи з цього, є можливість виділити наступні збірки (Рисунок 3.1).

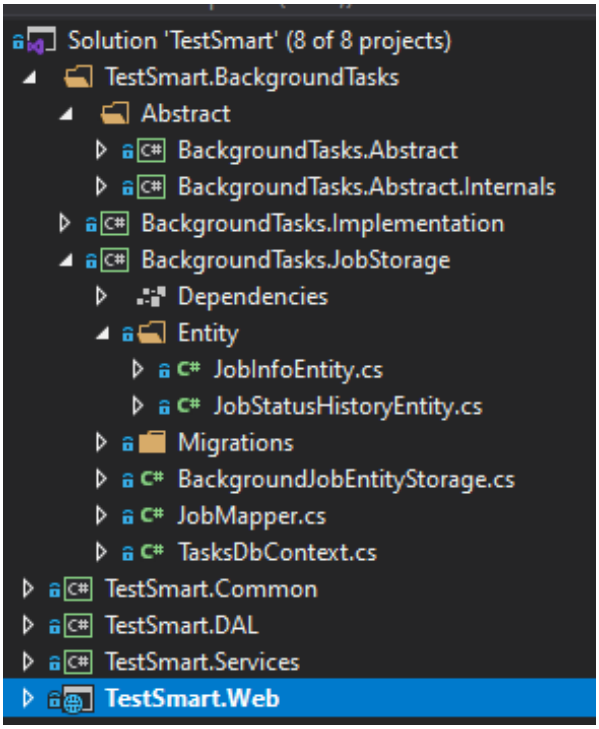

Рисунок 3.1. – Загальний вигляд веб додатка.

- BackgroundTasks.Abstract - зберігає абстракцію з описом логіки, що покриває всі функціональні вимоги до системи;
- BackgroundTasks.Abstract.Internals – зберігає абстракцію з опсом логіки та загальні конфігураційні файли, що відносять до механізму вирішення проблеми конкуруючих потоків;
- BackgroundTasks.Implementation - зберігають логіку реалізації конкретної поведінки механізму вирішення проблеми конкуруючих потоків;

- BackgroundTasks.JobStorage - збірка, що містить конкретні реалізації поведінок для роботи з MSSQL server за допомогою засобів ORM Entity framework збірка, що містить конкретні реалізації поведінок для роботи з MSSQL server за допомогою засобів ORM Entity framework. Також розміщається основна логіка управління даними задач з механізму конкуруючих потоків;
- TestSmart.Common - зберігає загальні класи, які використовується для допомоги в реалізації деякої функціональності, причому ця функціональність не є основним завданням програми;
- TestSmart.DAL – збірка, що містить конкретні реалізації поведінок для роботи з MSSQL server за допомогою засобів ORM Entity framework та з використанням підходу Code first. Тут, також, зберігаєм операціїї CRUD з джерела даних, оперуючи сутностями, які також знаходяться в даній збірці. TestSmartDbContext;
- TestSmart.Services - зберігання логіки реалізації конкретної поведінки по виконанню процесів системи. (В основному це сервіси, мапери та валідатори) ;
- TestSmart.Web – збірка, в якій зберігаєься інтерфейс мережевої взаємодії з системою що працює з абстракцією бізнес-логіки;

3.2. Структура репозиторію

Репозиторій – це патерн, який є проміжним між слоями доступу до даних та доменним рівнем. Працюючи, практично як in-memory колекція “доменних” об’єктів. Клієнти створюють декларативний опис запитів та передають ці запроси в репозиторій для виконання.

EntityFraemwork представляє для нас уже готове програмне рішення патерну Repository: DbSet<T> и UnitOfWork: DbContext. Проте, достатньо часто є

можливість бачити реалізацію яка є поверх існуючих в EntityFraemwork.(Рисунок 3.2).

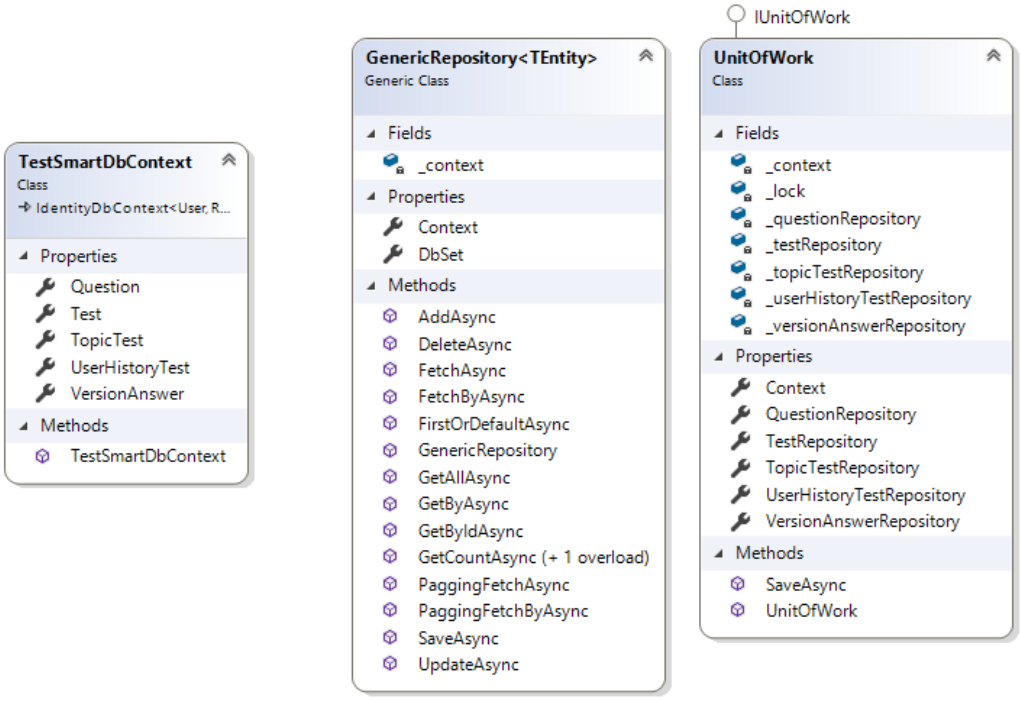

Рисунок 3.2. – Реалізація шаблони Repository та UnittOfWork.

Частіше за все використовують один із підходів:

1. Generic Repository, що виглядає, як спроба абстрагуватися від певного ORM;
2. Repository, як набір запитів до вибраної таблиці в базі даних;

В даному проекті використовують Generic Repository та UserRepository, TestRepository, UserHistoryTestRepository, RoleRepository, QuestionRepository, TopicTestRepository, VersionAnswerRepository:

- Generic Repository – виконує функцію загального репозиторія, який містить в собі повний список методів, які використовується в даній програмі. Дані методи дозволяють нам виконувати стандартний CRUD операцій, та містить додаткові методи для комфортної та полегшеної роботи. Поточний репозиторій є батьківським для всіх інших, які використовуються в даній програмі. Також, є можливість, сказати що

даний репозиторії представляє собою шаблон проектування, завдання якого – це управління доступом до даного джерела даних. Фактично TestSmartDbContext представляє собою реалізацію UnittOfWork – містить певну кількість репозиторіїв. Кожен з яких, представлений об'єктом DbSet, який надає нам функціональність, за якої є можливість отримувати, додавати, видаляти дані. Потрібно додати, що даний репозиторій зберігає в собі посилання на визначений контекст та набір DbSet, що потрібен для роботи з поточної сутністю. Всі методи сховища фактично викликають методи DbSet і контексту даних. Варто відзначити, як відбувається завантаження пов'язаних даних. Якщо у нас навігаційні властивості позначені як віртуальні, то за допомогою Lazy Loading пов'язані дані автоматично будуть довантажуватися до завантажених сутностей;

- UserRepository – репозиторій, що містить конкретну реалізацію Generic Repository. Зроблене наслідування від IGeneric Repository, після чого реалізовується будь-які методи, що визначені в IUserRepository;
- TestRepository – дана реалізація містить в собі основний CRUD операцій та додаткові методи, які в свою чергу працюють з табличкою в базі даних. Саме за допомогою цього, є можливість добавляти, видаляти, зберігати та обновляти дані тести. Також використовують методи, що в свою чергу містять запити IQueryable;
- Інші репозиторії працюють аналогічно, що дає можливість використовувати CRUD операцій для кожної окремої сутності (Рисунок 3.4);

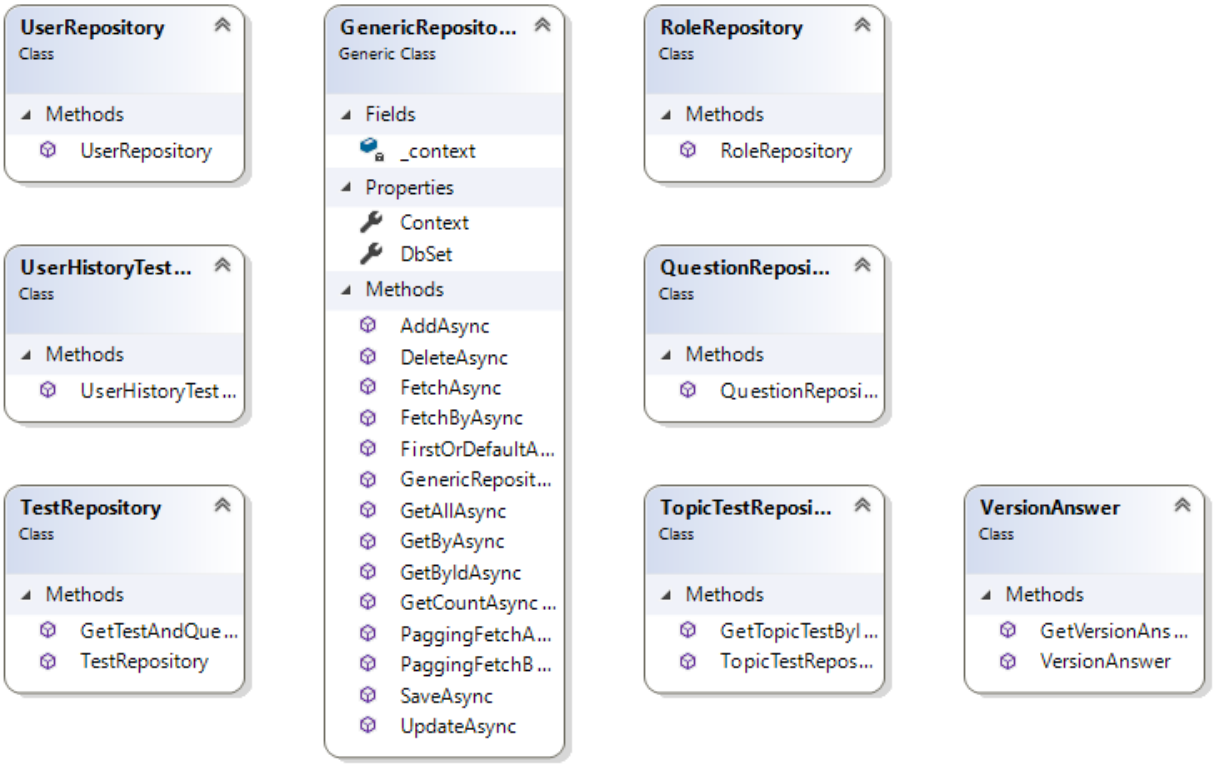

Рисунок 3.4. – Реалізація шаблону Repository для всіх сутностей.

IQueryable в свою чергу представляє віддалений доступ до бази даних а також дозволяє нам переміщатися по даним як в порядку від початку до кінця, так і від кінця в початок. Під час процесу, коли створюють запит, об’єктом що повертає IQueryable, відбувається оптимізація запиту. В кінці в процесі його виконання витрачають менше пам’яті, а також менше пропускної здатності мережі, проте одночасно він може оброблятись трішки повільніше ніж звичайний запит, який повертає об'єкт IEnumerable.

3.3. Реалізація механізму конкуруючих потоків

Вході роботи механізму оцінювання тесту та специфікою роботи Web API випливає наступна проблема. Оскільки є вірогідність конкуруючих потоків обновлять спільні набори даних в один момент, у цьому випадку оцінка тесту могла б видати не достовірний результат проходження тесту. В

ході виникнення поточної проблеми було розроблено механізм. Перш за все було створено база даних, для створення черги послідовних виконання задач, щоб не було накладення ваги питання до відповідного тесту.

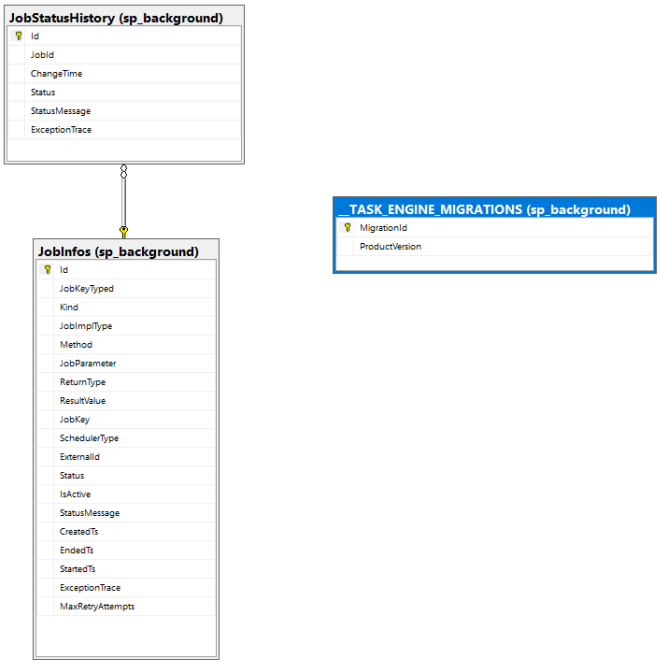

Рисунок 3.4. – Структура бази даних про виконання задач.

Як бачимо з схеми бази даних в таблиці JobInfos, що на рисунку 3.4, є вся інформацію про задачу. Це параметри які приймає та повертає відповідний метод, поточний статус виконання задачі, дати створення та закінчення задачі. Таблиця JobStatusHistory виконує роль збереження історії про виконання поточної задачі. Це як статуси виконання, час виконання та відповідні повідомлення кожної задачі.

Для зручної роботи з даними про задачі було створено абстракції, щоб обмежити нас від конкретного джерела даних. Дана абстракція це інтерфейс IPersistedBackgroundJobStorage та відповідно його реалізація BackgroundJobEntityStorage.

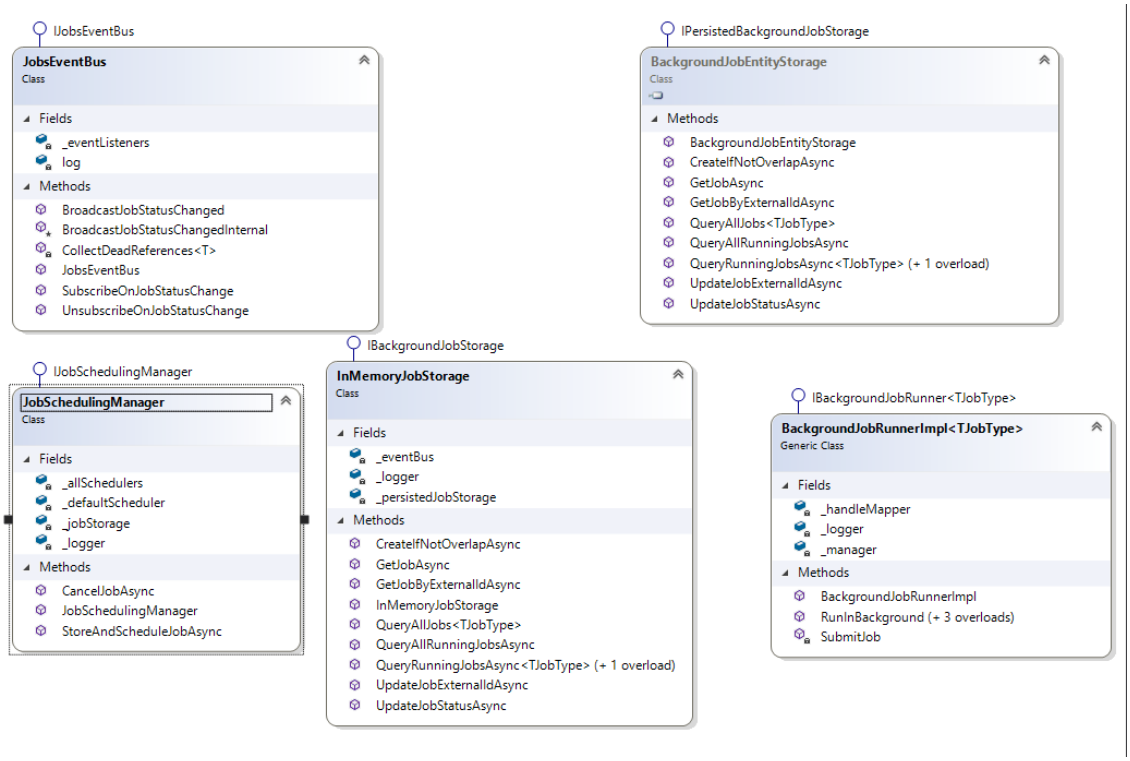

Рисунок 3.5. – Діагрма класів по збереженню даних про задачу.

Реалізацію класу BackgroundJobEntityStorage використовує додаткова обгортака InMemoryJobStorage. Клас InMemoryJobStorage перед тим передати дані до даної бази даних, виділяє додатковий поток який працює з бібліотекою Hangfire через клас JobSchedulingManager (Рисунок 3.5).

Виходячи з вище сказаного мають клас InMemoryJobStorage який працює з бібліотекою Hangfire та даною базою даних за допомогою додаткових абстракцій. Більш детальна робота цього класу в наступному розділі.

Основний клас механізму, який на далі використовується для роботи з конкуруючими потоками. Це клас BackgroundJobRunnerImpl, який звісно є абстрагований інтерфейсом IBackgroundJobRunner<TJobType> де додатково передається шаблон TJobType, це потрібно для розуміння який тип задачі хочеться виконувати. В конкретній реалізації для перерозподілу ваги питання це enum JobTypeEnum. Дальше, опис конкретної реалізації класу BackgroundJobRunnerImpl.

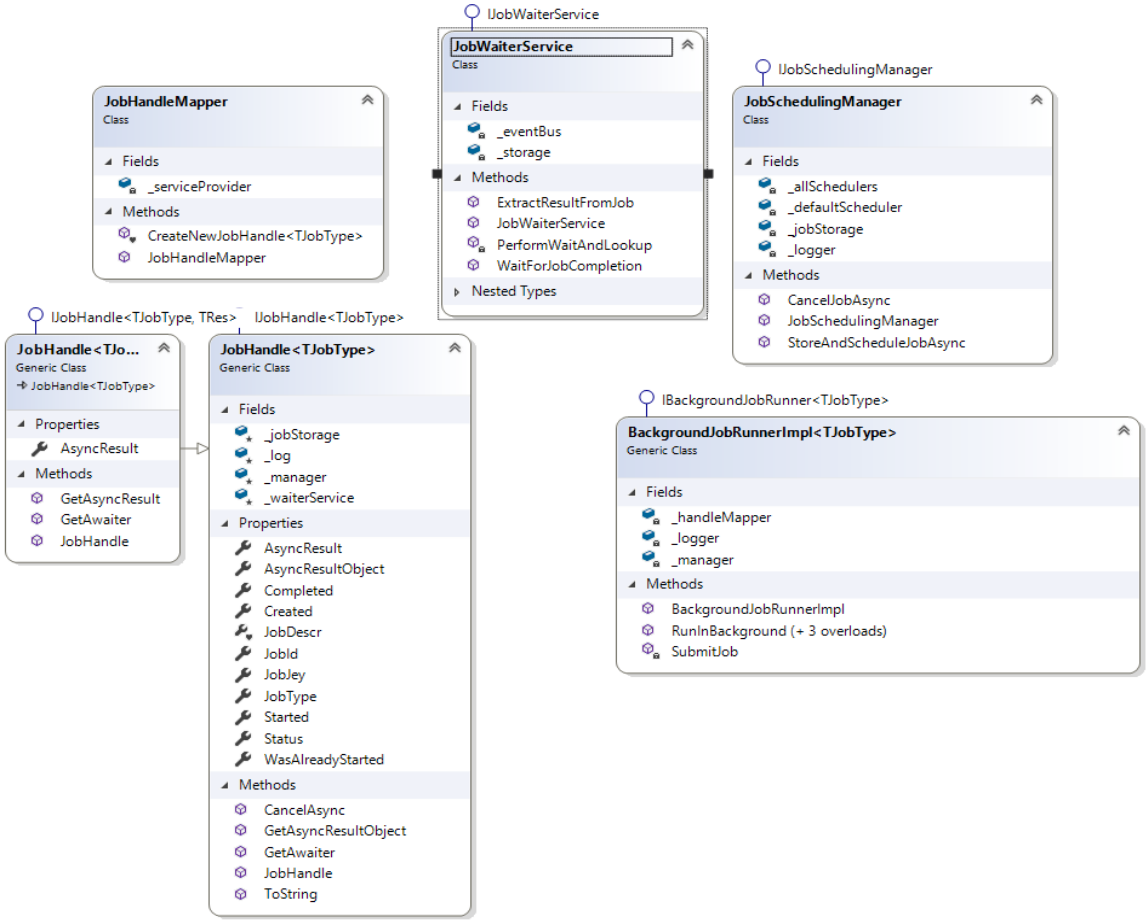

Рисунок 3.6. – Діаграма класів по роботі з даними про задачу.

Як видно з рисунку 3.6 клас BackgroundJobRunnerImpl має основний метод RunInBackground та його три перезавантаженні версії, які приймаю різні параметри та надають змогу більш гнучкіше працювати з поточним механізмом. Особливо таку можливість надає перезавантажений метод який приймає Expression<Func<TJobInstance, Task>> voidCallAction, який надає змогу примати буть який асинхронний метод в даній API. Всі перезавантаженні методи вертають об’єкт JobHandle, що надає змогу управляти результатом асинхронної задачі, та отримувати відповідні статуси виконання задачі. Оскільки методи RunInBackground можуть приймати різні дані, нам ці дані потрібно впорядкувати відповідно до інтерфейсу IJobSchedulingManager (який був описаний вище). Для цього був створений додатковий клас JobHandleMapper, який вміє це робити.

Весь поточний механізм що описаний, тримається на абстракції без конкретних класів. Для цього нам потрібно деякий розширений клас, який буде створювати відповідність між інтерфейсом та його конкретної реалізацією. Це метод AddBackgroundTaskEngine який використовує стандартну механізм IoC контейнер.

Тепер, до самого використання реалізації інтерфейсу IBackgroundJobManager. Для цього створюється клас GetAndUpdateTestJob який є реалізацією інтерфейсу IBackgroundJob. В якого є єдиний метод Execute, який буде записувати до бази даних і виконуватися по черзі не залежно на якому потоці буде створено, що є вирішенням даної проблеми. Тип клас GetAndUpdateTestJob передають до метода RunInBackground, яка в сою чергу буде виконувати механізм конкуруючих потоків.

3.4. Реалізація планувальника задач

Як було описано в реалізації конкуруючих потоків, нам потрібен планувальник задач, який буде забезпечувати збереження задачі на обновлення ваги питань до відповідного тесту. Як було розглянуто вище, одним з найкращих варіантів по вибору планувальника задач є бібліотека Hangfire, але в поточній реалізації абстрагувалися від конкретного планувальника задач за допомогою загального інтерфейсу – IjobScheduler. В даному інтерфейсу виділили основні методи, які допоможуть нам в реалізації конкуруючих потоків. Перший метод це ScheduleItemAsync, який приймає об’єкт класу JobDescriptorNonTypeSafe, що описує задачу від назву метода який потрібно буде виконанти до статусу виконання задачі. Другий метод інтерфейсу це CancelJob який по індексатору задачі може відміти її. Поточні описані класи зображені далі на діаграмі класів. Поточною реалізацію інтерфейсу IjobScheduler є HangfireJobScheduler.

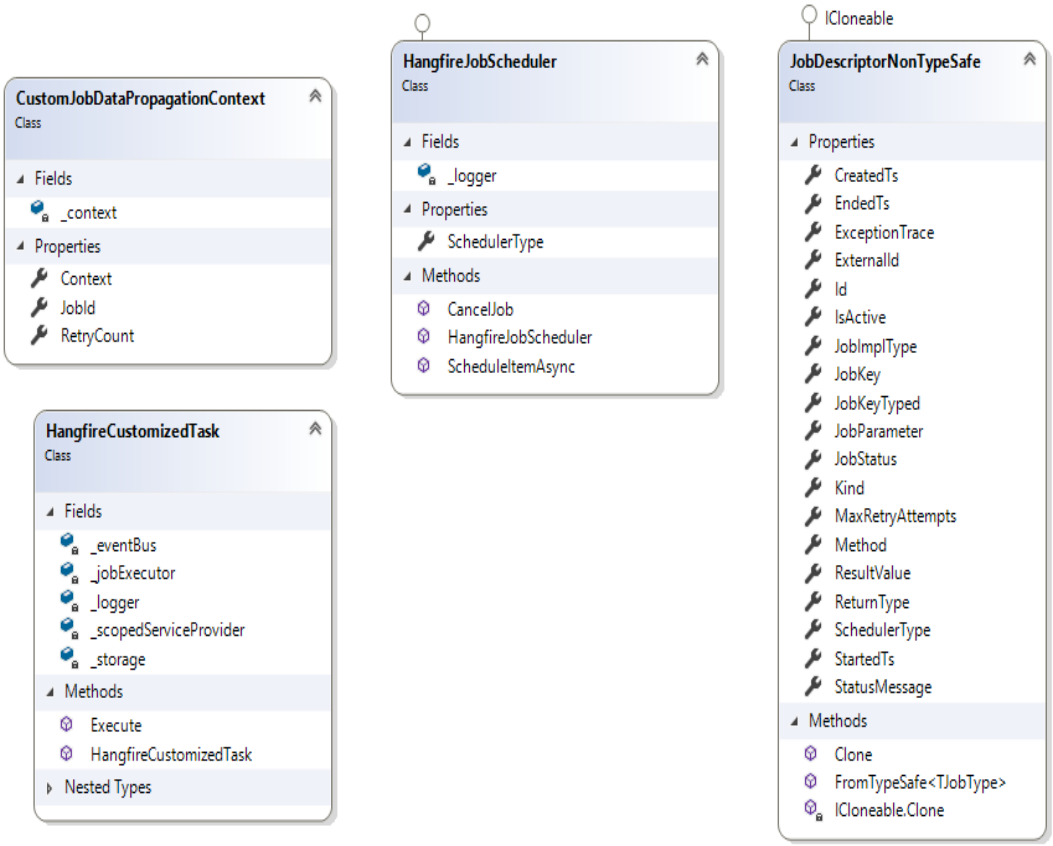

Рисунок 3.7. – Діаграма класів по роботі з планувальником задач.

В даному випадку поточна реалізація винесена до окремої зборки, що надає змогу гнучкої її використовувати під різні типи задач. В даному випадку одна з таких задач оптимізоване обновлення ваги питання до тесту (Рисунок 3.7).

Також одним з важливих плюсів бібліотеки Hangfire є зручний інтерфейс для аналітики виконаних задач. На основі цих даних є можливість сказати скільки раз і який тест був обновлений. Наприклад це видно з наступного аналізу.

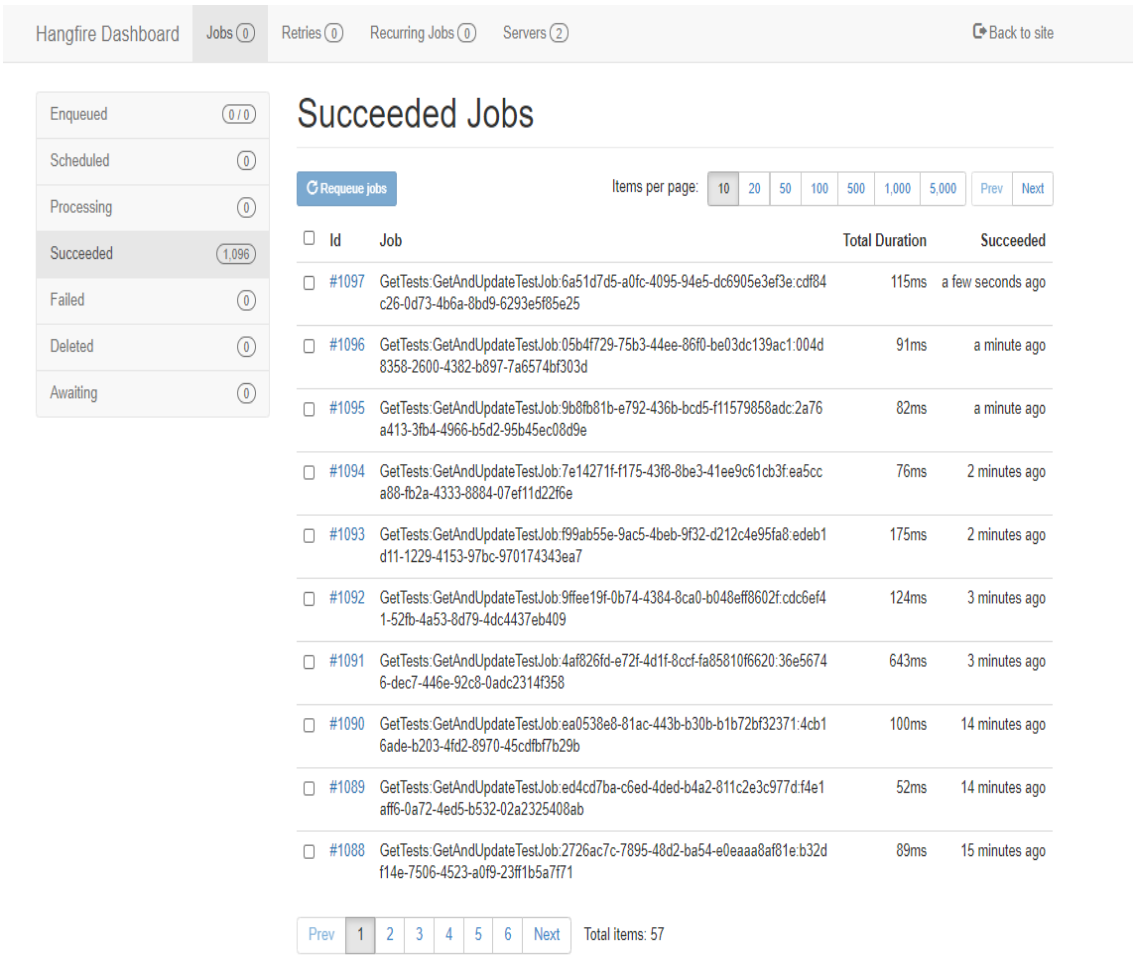

Рисунок 3.8 – Успішні виконані задачі планувальником.

Виконаних обновлень питань становить 1096 і 0 не успішних (Рисунок 3.8). З поточного рисунку є можливість побачити скільки і коли виконувався запит. У випадку не успішного виконання задачі інтерфейс дозволяє нам його перезапустити, що надає гнучкість системі.

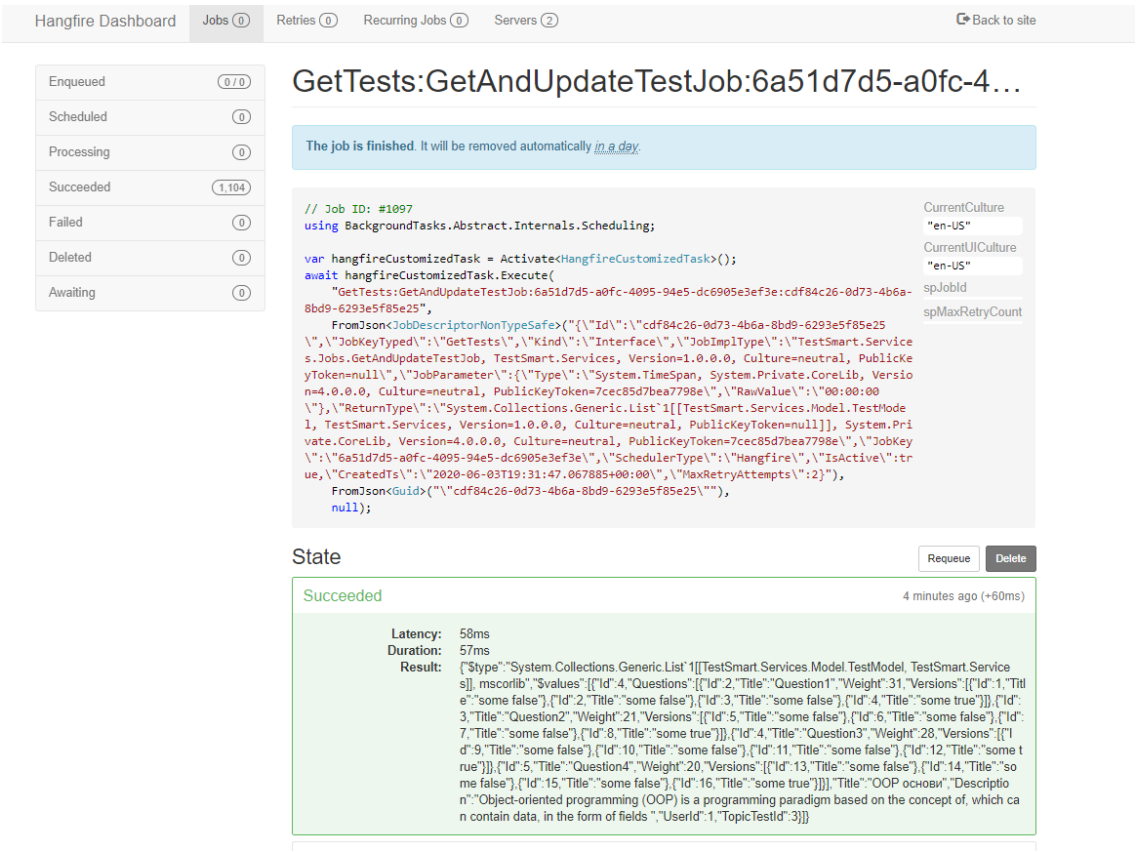

Рисунок 3.9. – Виконана успішна задача.

На рисунку 3.9. зображено параметри задачі, яка була поставлена та успішно виконана. В даній реалізації всі задачі, які є можливість додати до планувальника, повинні реалізовувати інтерфейс IBackgroundJob<in TParameter, TResult> в якого тільки один метод Execute. Але за допомогою є шаблонів які приймає інтерфейс метод стає більш гнучкішим і абстрагованим від сигнатури методу.Для задачі обновлення даних по перерозподілу ваги питання є реалізація класу GetAndUpdateTestJob в якого метод Execute вертає список обновлених тестів, що працює через інтерфейс IUnitOfWork, що було описано вище. Основна задача метода це отримати всі запитання тесту, перерахувати вагу питання та обновити дані в базі даних В результаті чого отримують обновлені дані про тести та інформацію про кожні із запитань.

Для збережені всієї інформації, що є можливість бачити на користувацькому інтерфейсу потрібно зберігати інформацію в джерелі даних. В поточній реалізації вибрали базу даних MS SQL.

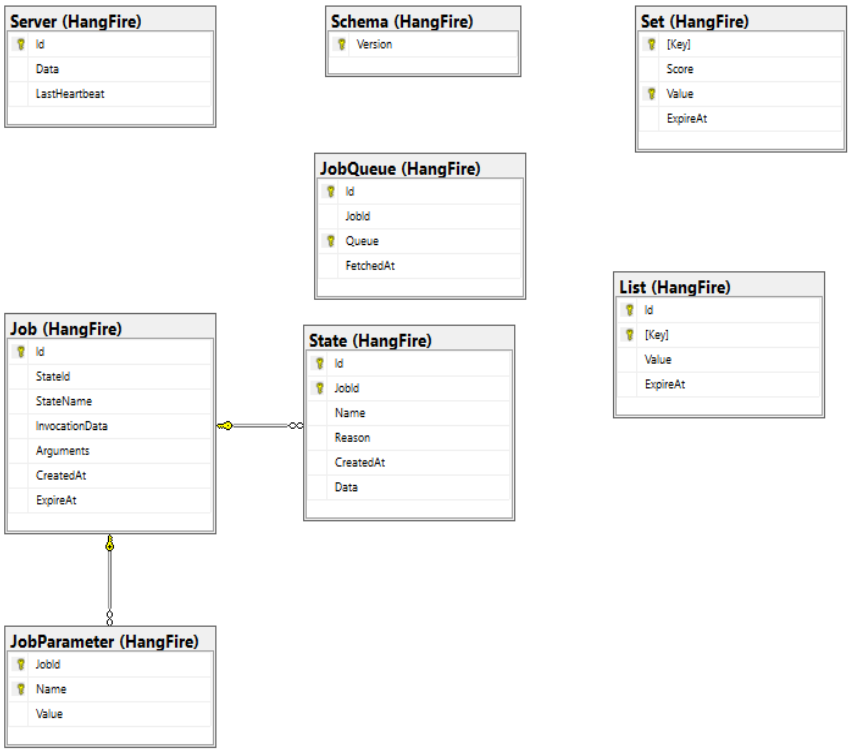

Рисунок 3.10. – Структура бази даних Hangfire.

Як видно з структури бази даних Hangfire центральною таблицею є Job, в якій зберігається все про задачу яку потрібно виконати (Рисунок 3.10). Також є допоміжні таблиці які виконують роль додаткової інформації про задачі, наприклад таблиця State яка зберігає дату створення, дані які мають повертатися з виконаної задачі.

3.5. Алгоритм перерозподілу ваги питання

Вході виконання роботи було розроблено механізм ранжування для окремого тесту. На основі відповідей на питання у проходженні будь-якого тесту кожного з користувачів, будується відношення оцінки для кожного питання в тесті. При закінчені тестування, Web API створює задачу, яка має обновляти кількість правильних чи не правильних відповідей на питання. Після того як дані обновилися, тобто запланована задача API успішно виконалися, відбувається перерозподіл ваги між питаннями відповідного до

поточного тесту. В даному механізмі враховуються кількість не правильних відповідей на одне запитання. Для знаходження ваги тесту користувалися наступною формулою:

$$w = \frac{100 * Qm}{\sum_0^Q Qm},$$

де w - це вага за питання, Qm – це кількість не вірних відповідей для одного запитання, Q – це кількість питань в тесті.

В такому випадку виникає наступна вірогідність, що вага питання може прямувати до нуля, що є критичним в даній системі. Для рішення цієї проблеми було зроблено обмеження, яке корегується наступною формулою:

$$w_{min} = \frac{100}{\sqrt[2]{Q^3}},$$

де w_{min} - це мінімальна вага за питання, Q – це кількість питань в одному тесті. Таким чином вирішили проблему з мінімальною вагою відносно кількістю питань в тесті.

Основним класом для виконання алгоритмом перерозподілу ваги у веб застосунку є Recalculation, основним методами поточно класа є:

- GetMinWeight основна задача метода полягає перерахунок мінімальної ваги для питання в межах поточного тесту. Метод на вході приймає кількість питань, які є актуальними для тесту, що отримує користувач, відповідно на виході метод GetMinWeight вертає мінімальну вагу для питання;
- GetWeights метод вертає масив рівно розподілених ваг для кожного з питань поточного тесту, керуючись на кількість питань. Потрібний для розподілення ваг, коли тест тільки створився;
- RecalculationWeight основний метод який виконує задачу перерозподілу кожного питання в поточному тесту;

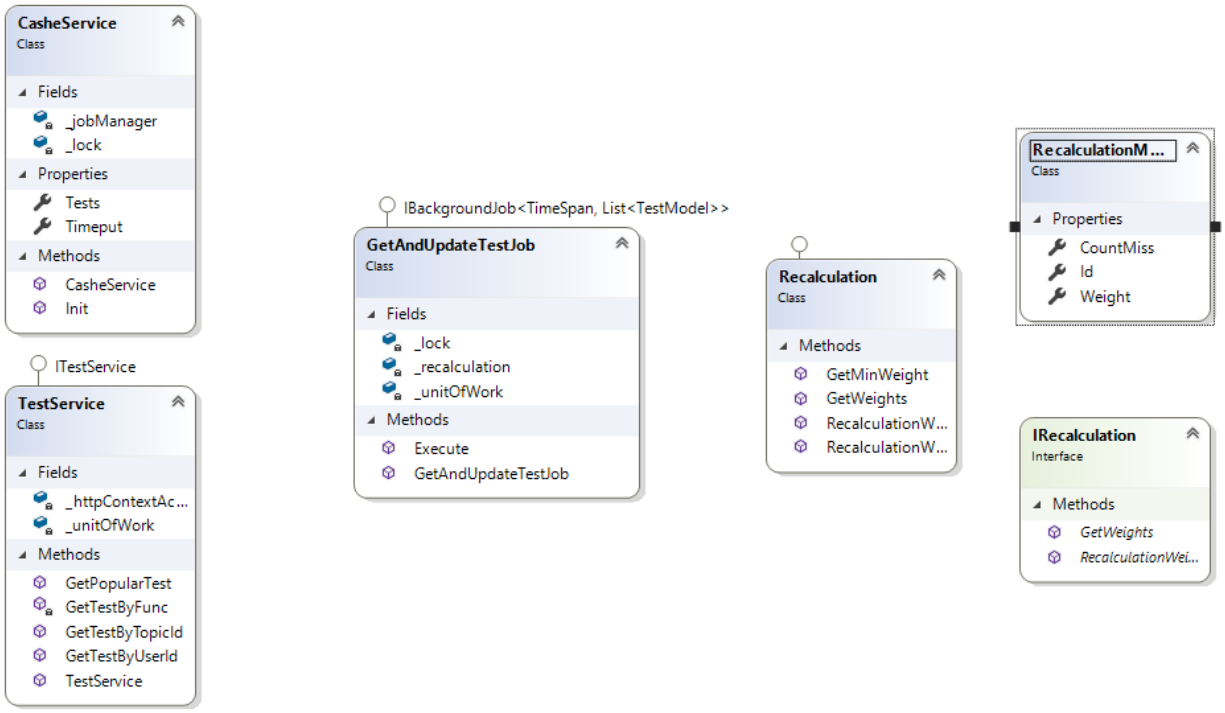

Рисунок 3.11. - Діаграма класів механізму перерозподілу ваги питання.

Як видно з діаграми класів на рисунку 3.11, клас Recalculation відповідає за логіку перерозподілу ваги тести та має основні для цього методи. Більш детально метод RecalculationWeight, який приймає об'єкт класу RecalculationModel, що містить в собі інформацію про індикатор питання, кількість помилок які допустили користувачі, та відповідно саму вагу питання, яку в процесі виконання цього методу перераховується. З цього слідує, що повинно вернути з даного методу, список перерозподілених ваги питання.

3.6. Механізм авторизації

В реалізації даного веб додатка, потрібно надати можливість користувачу авторизуватися. Як було розглянуто в 2 розділі про типи аутифікації користувачі і зроблений висновок, що для дан:ої системи по всім критеріям було обрано JWT-токен. В даному програмному продукту JWT-токен реалізація наступна.

Перш за все було створена таблиця User та відповідно клас User. Також додали в проект спеціальний клас AuthOptions, який буде описувати ряд якостей, необхідних для генерації токен.

```
public class AuthOptions
{
    public const string ISSUER = "MyAuthServer";
    public const string AUDIENCE = "MyAuthClient";
    const string KEY = "mysupersecret_secretkey!123";
    public const int LIFETIME = 1;
    3 references | 0 exceptions
    public static SymmetricSecurityKey GetSymmetricSecurityKey()
    {
        return new SymmetricSecurityKey(Encoding.ASCII.GetBytes(KEY));
    }
}
```

Рисунок 3.12. – Структура класу AuthOptions.

Константа ISSUER (Рисунок 3.12) представляє видавця токена. Тут можна визначити будь-яку назву. AUDIENCE представляє споживача токена - знову ж таки може бути будь-який рядок, але в даному випадку адреса був призначений додатковий поточного додатка. Константа KEY зберігає ключ, який буде застосовуватися для створення токена. Для вбудовування функціональності JWT-токенів в конвеєр обробки запиту використовується компонент JwtBearerAuthenticationMiddleware.

Для обробки запиту в контролері створений метод Token, який зіставлений з маршрутом "/ token". Цей метод обробляє запити POST і через параметри приймає логін і пароль користувача.

Сам токен являє об'єкт JwtSecurityToken, для ініціалізації якого застосовуються всі ті ж константи і ключ безпеки, які визначені в класі AuthOptions і які використовувалися в класі Startup для настройки JwtBearerAuthenticationMiddleware. Важливо, щоб ці значення збігалися.

3.7. Загальна структура API

В даній розробці Web API виділили наступні критерії для користувача:

Авторизуватись і зареєструватись. Інтерфейс API надає кінцеві точки для авторизації і реєстрації користувача в системі для цього необхідні наступні дані: email, пароль.

Відновити пароль Інтерфейс API повинна мати інтерфейс користувача для відновлення пароля. На першому етапі API має запросити введення email, потім перевірити його на коректність і присутність в джерелі даних, після чого відправити на введений email посилання для скидання пароля.

Задати критерії пошуку тесту:

- вибрати (під) тип тесту;
- вибрати тест;
- вибрати продукт;

API повинна мати зручний вибір критеріїв. Особливість вибору полягає в тому, що для початку необхідно вибрати тип, після чого тест. Після виконання цих дій буде запропонований список питань.

Отримати результати тестуванню. Користувач має змогу продивитися на результат кожного з запитань, на які користувач мав змогу відвідати. Також у користувача системи має бути історія тестувань і його результатів відповідно до його проходження тестів за певний період.

Згідно з цими критеріями було розроблено API та для його тестуання була використана бібліотеки Swagger, яка зображена на наступному рисунку.

Як видно з рисунку 3.13. реалізовано основні методи REST API, а саме:

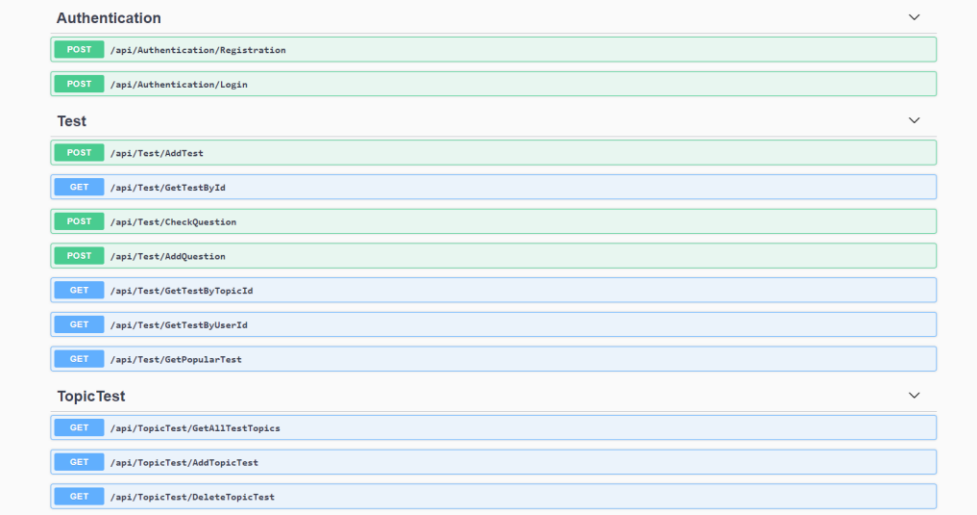

Рисунок 3.13. – Методи API.

- api/Authentication/Registration і api/Authentication/Login – POST методи, що виконують процес авторизації та реєстрації, що надає клієнту зручне використання API інтерфейсу, не прив'язуючись до конкретної технології;
- api/Test/AddTest і api/Test/AddQuestion – POST методи які дозволяє користувачу створювати нові тести і відпідно до поточного створеного тесту додавати нові запитання;
- api/Test/CheckQuestion – POST метод що приймає список пройдених запитань відповідно до поточного тесту конкретним користувач. В свою чергу метод вертає результат проходження тесту;
- api/Test/GetTestById, api/Test/GetTestByTopicId – GET методи, що повертаю тести до відповідного заданого Id клієнтської сторони;
- api/TopicTest/GetAllTestTopics, api/TopicTest/AddTopicTest – GET методи, що дозволяються працювати над темами тестів;

Як видно з рисунка 3.14. метод api/TopicTest/AddTopicTest, вертає інформацію про теми тестів у JSON формат.

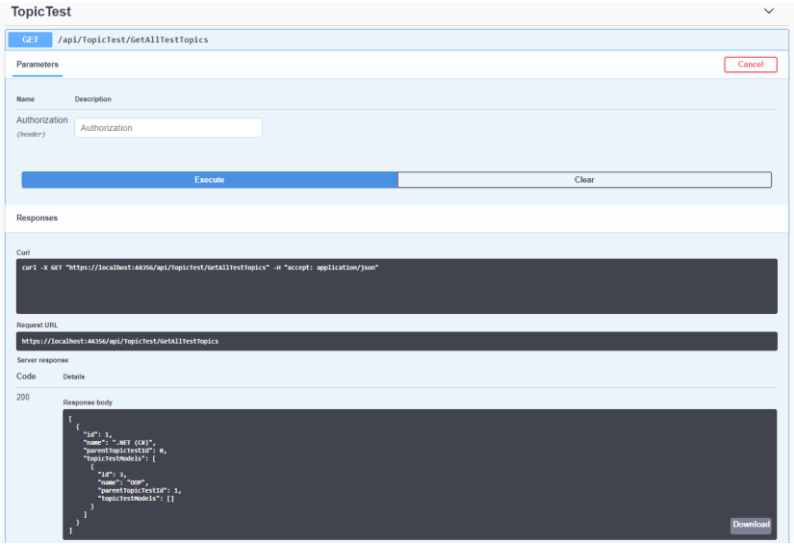

Рисунок 3.14. – Використання методу GetAllTestTopics за допомогою Swagger.

Відповідно до кожної теми може бути дочірні теми. Якщо користувач не авторизувався то сервер вертає 401 помилку, що значить користувач не авторизований і немає достатньо прав. У випаду, помилки на сервері, API вертає 500 помилку зі статусом про неуспішність.

3.8. Публікація API на платформу Azure

Одним з головних етапів розробки є публікація веб додатка на сервер, для доступу користувачів в мережі Інтернет, та для розробки клієнтської частин, що є індивідуальним частиною завданням.

Перше за все потрібно створити бази даних на платформі Azure та отримати відповідні рядки для з'єднань до відповідної бази даних.

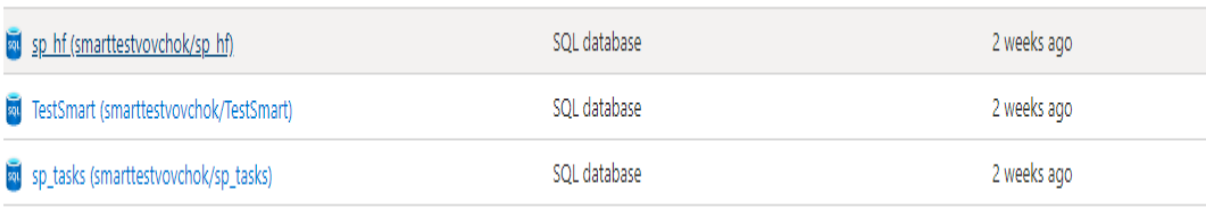

Рисунок 3.15. – Список баз даних на платформі Azure.

Кожну базу даних розмістили на платформі Azure та отримали відповідні строки для з’єднань (Рисунок 3.15):

- "DefaultConnection":"Server=tcp:smarttestvovchok.database.windows.net,1433;Initial Catalog=TestSmart";
- "SpTaskEngineConnection":"Server=tcp:smarttestvovchok.database.windows.net,1433;Initial Catalog=sp_tasks";
- "HangfireConnection":"Server=tcp:smarttestvovchok.database.windows.net,1433;Initial Catalog=sp_hf;Persist";

Після публікації баз даних, нам потрібно опублікувати сам веб додаток, та отримати посилання вже на веб сайт. Дана посилання https://testsmartweb20200606114325.azurewebsites.net/swagger/index.html веде на сторінку swagger, що надає змогу зручно тестувати веб додаток клієнтської стороні індивідуального завдання.

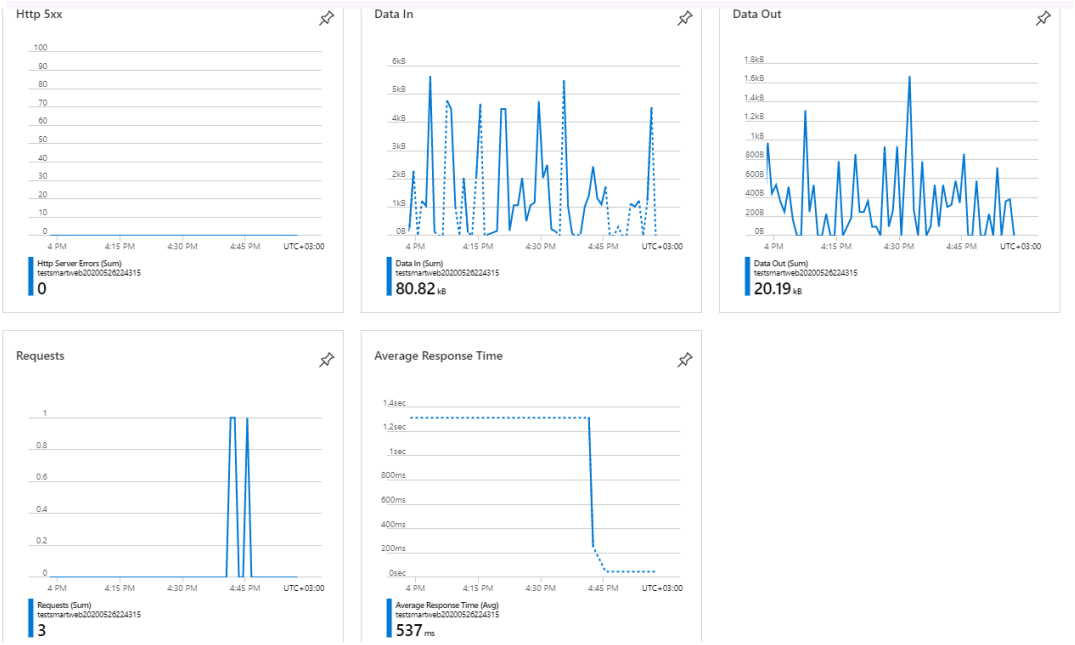

Рисунок 3.16. – Графіки роботи веб додатка.

Дані графіки, що зображені на рисунку 3.16, надають змогу досліджувати скачки і провали в значеннях метрик. Використовуйте оглядач метрик для перевірки працездатності і використання даних ресурсів.

ВИСНОВОК

Під час виконання дипломного проєкту спроектовано індивідуальну частину системи автоматизованого тестування на основі відносного оцінювання по статистичним вибіркам та розроблено загальні механізми між модульної взаємодії та реалізовано шаблонну поведінку відповідних модулів.

У ході огляду існуючих рішень було розглянуто найпопулярніші веб-орієнтовані сервіси для тестуванню та описано їх переваги та недоліки. Встановили відповідні завдання до сервісної частини.

Проведено аналіз предметної області, в результаті чого визначено основні вимоги до формату тестів, що генеруються системою. У ході огляду існуючих рішень було розглянуто найпопулярніші веб-орієнтовані системи тестування, визначено їх переваги та недоліки, встановлено вимоги до серверної частини.

Архітектурним рішенням для системи став клієнт-серверний додаток. Розглянули найпопулярніші інструменти для створення веб-додатків, а саме Spring (Java), ASP.NET Core (C#) , GGI (C++). Технічним рішенням для сервісного частини є побудова серверу за допомогою технології ASP.NET Core мовою програмування C# при цьому використовували бібліотеки та фраємворки, такі як Hangfire, Entity Framework Core. Було розглянуто недоліки інших бібліотек і фраємворки, а саме ADO.NET, Quartz.NET та інші.

У ході роботи було розроблено механізм перерозподілу ваги, який виконує основну роботу даної системи. Було розроблений неменше важливий механізм запису даних ваги питання з різних потоків.

Описано розподілена архітектура між компонентами бізнес-логіки та їх взаємодію. Використання доцільних шаблоних рішень в даному веб додатку, який гнучким для подальших зміни та додавання нового функціонала.

Розроблено формат взаємодії між клієнтською та серверною частиною, вимоги до REST API серверу, та створено зручний інтерфейс для тестування

веб додатку за допомогою бібліотеки Swagger. Вирішено ряд проблем з ключовими механізмами з розподілу ваги питання у тесту. Визначено вимоги до швидкодії та відмово стійкості системи.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Леон Шкляр, Річ Розен., книга «Архітектура веб - додатків», рік видання: 2011 року
2. Грамотна клієнт-серверна архітектура: як правильно проектувати і розробляти web API [Електронний ресурс] Доступ: https://echo.lviv.ua/dev/6455
3. Сервис, архитектура, управление и бизнес-термины [Електронний ресурс] Доступ: https://www.ibm.com/developerworks/ru/library/ws-soa-term1/index.html
4. Hangfire — планировщик задач для .NET [Електронний ресурс] Доступ: https://habr.com/ru/post/280732/
5. Производительность кода и требовательность к ресурсам [Електронний ресурс] Доступ: https://habr.com/en/post/262461/
6. Web Framework Benchmarks [Електронний ресурс] Доступ: https://www.techempower.com/benchmarks
7. Entity Framework 6 [Електронний ресурс] Доступ: https://docs.microsoft.com/en-us/ef/ef6/
8. Метрики качества ранжирования [Електронний ресурс] Доступ: https://habr.com/ru/company/econtenta/blog/303458/
9. Hangfire [Електронний ресурс] Доступ: https://www.hangfire.io/
10. Quartz.NET – Quartz Enterprise Scheduler .NET [Електронний ресурс] Доступ: https://www.quartz-scheduler.net/
11. Моделі і методи швидкого створення веб застосувань [Електронний ресурс] Доступ: https://ela.kpi.ua/bitstream/123456789/27860/1/Vovk_magistr.pdf.
12. Джеффри Рихтер CLR via C# / Рихтер Джеффри. – Санкт-Петербург: Питер, 2011. – (ISBN 978-5-7502-0348-2).

13. Учим программированию [Електронний ресурс] Доступ: https://geekbrains.ru/.
14. Тестування знань [Електронний ресурс] Доступ: http://www.quizful.net/test